
175 PTAS
89
mi computer
CURSO PRACTICO DEL ORDENADOR PERSONAL
EL MICRO Y EL MINIORDENADOR
The Banana Interface
INPUTS
OUTPUTS
+12v LOGIC OUTPUTS
Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen VIII-Fascículo 89

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones
Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°, 08021 Barcelona
Tel. (93) 200 19 02

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-067-2 (tomo 7)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 098510
Impreso en España-Printed in Spain-Octubre 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

No se efectúan envíos contra reembolso.



Scout y B-Star

Continuando con nuestro estudio de la planificación estratégica en AI, examinaremos algunas estrategias alternativas para juegos de azar

El procedimiento *alfa-beta*, que explicamos en el capítulo anterior, representa un gran avance sobre la minimaximización directa (puesto que identifica y suprime del árbol del juego las ramificaciones redundantes); durante muchos años ha sido el punto central de los programas de ajedrez por ordenador más eficaces. Pero recientemente se han propuesto dos estrategias alternativas. Una es el *algoritmo Scout*, de Judea Pearl; la otra, el *algoritmo B-Star (B*)*, de Hans Berliner.

La esencia del método Scout consiste en disponer de una función de evaluación sumamente afinada que se pueda utilizar para rechazar, sin más búsqueda, los movimientos poco plausibles. Sólo es necesario examinar en profundidad los movimientos que parecen más prometedores.

El método B* escudriña los movimientos del nivel superior del árbol e intenta, lo más rápidamente posible, cumplir uno de estos dos cometidos:

- Demostrar que el movimiento *aparentemente* mejor es en realidad el mejor disponible.
- Demostrar que ninguno de los movimientos alternativos es mejor que otro.

Esta estrategia gemela se implementa mediante un par de procedimientos denominados ProveBest (demostrar mejor) y RefuteBest (refutar mejor), que se basan en la asignación de los valores a cada nudo del árbol: uno, una evaluación optimista; otro, una pesimista. El objetivo es obligar al algoritmo de búsqueda a concentrarse en áreas del árbol del juego en donde hay incertidumbre y donde esta falta de certeza podría incidir en la decisión final.

Sería equivocado suponer que la búsqueda arborescente es el único enfoque a los juegos por ordenador. Existen algunos juegos interesantes en los que la filosofía de búsqueda parece no dar buenos resultados, independientemente de los ingeniosos trucos que se adopten para perfilar el proceso. Entre éstos se incluyen muchos de los juegos de naipes más populares (en particular el bridge y el poker) y varios juegos de tablero (como el *go* y el *go-moku*).

Estos juegos se pueden practicar con diversos niveles de destreza, y se dice merecidamente que son los más sobresalientes exponentes de juegos inteligentes. Pero todos los intentos por programarlos de acuerdo al esquema de la búsqueda arborescente han tropezado con obstáculos inesperados. Una

Búsqueda vana
Aunque los ordenadores pueden jugar inteligentemente a algunos juegos empleando métodos de búsqueda para anticiparse a un cierto número de movimientos, hay muchos juegos en los que las estrategias de búsqueda no son eficaces; ello se debe a que el juego posee un elemento de azar, como el *backgammon*, o bien a que el árbol del juego se bifurca rápidamente en sucesivos movimientos para producir una ingente cantidad de posibles permutaciones de movimientos. Para crear programas que puedan jugar a esta clase de juegos, se deben procurar métodos alternativos

David Lawrence

Terminología de la búsqueda arborescente

Árbol de juego	Estructura arborescente que se forma considerando los movimientos posibles, seguidos por las posibles réplicas del adversario, etc.
Nivel	Un nivel del árbol de juego.
Anticipación	Proceso de construcción de un árbol de juego.
Valor de un nudo	El valor asignado a un nudo del árbol de juego mediante el examen de los valores por debajo de él, trabajando desde abajo.
Minimaximización	Elección del valor que sustentará el árbol mediante la minimización en los niveles impares (mueve el otro) y la maximización en los pares (mueve uno).
Algoritmo "alfa-beta"	Refinamiento de la minimaximización que elimina las porciones del árbol de juego que se estima no pueden tener incidencia en el nivel superior.
Factor de bifurcación	Cantidad media de bifurcaciones o movimientos en cada nivel del árbol de juego. El *go*, juego oriental, posee un factor de bifurcación de más de 200.

razón es que el factor de bifurcación es demasiado amplio, generando una cantidad tan enorme de posibles combinaciones de movimientos que el ordenador no puede manipularlas todas al mismo tiempo.

Una respuesta más profunda es que los algoritmos de búsqueda arborescente son una burda aproximación a la forma en que el jugador humano aborda el problema: los expertos buscan sólo mentalmente en sus propios árboles de juego, construidos internamente en circunstancias especiales, e incluso así sin demasiada eficacia.

Hans Berliner, quien ideó el método B*, desarrolló un programa de *backgammon* que en 1980, en una partida de desafío, venció al campeón del mundo. Pero el programa no efectúa ninguna búsqueda en absoluto, al menos en el sentido convencional. Si usted piensa en cómo se juega una partida de *backgammon*, comprenderá que su árbol de juego es "probabilístico": el papel de los dados introduce bifurcaciones que están bajo el control de cualquiera de los jugadores, lo que dificulta la implementación de muchos de los procedimientos de anticipación esbozados previamente.

En lo que el programa de Berliner es muy bueno (mucho mejor que nosotros) es en calcular las probabilidades de que se produzcan durante el juego diversas combinaciones de dados y fichas. Posee, asimismo, una función de evaluación muy sofisticada. De hecho, en realidad emplea varias funciones para diferentes fases del juego y varía constantemente entre ellas a medida que el juego va progresando.

El *go* es un juego oriental, que se juega desplazando piezas sobre una cuadrícula de 18 por 18, con el objeto de rodear áreas de la cuadrícula para ganar territorio, y de rodear las piezas del oponente para eliminarlas del juego. Carece de un elemento de azar, pero el factor de bifurcación es tan inmenso que las técnicas basadas en búsqueda resultan infructuosas.

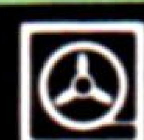
Los mejores programas de *go* perciben el tablero en términos de unidades mayores que meras piezas individuales (tales como *cadenas* y *ejércitos*) que son agrupamientos significativos para el ojo humano; una de las razones por las cuales la programación del *go* está más atrasada que la del ajedrez puede que sea que nuestra comprensión de la percepción humana es inadecuada. Quizá los japoneses, que veneran el juego, consideren al *go* como un proyecto adecuado para sus máquinas paralelas de la quinta generación. Ciertamente, un programa que juegue al *go* con éxito desarrollará el concepto de inteligencia artificial hasta sus límites.

Ya existen programas de *go* para los actuales ordenadores personales, y próximamente consideraremos los problemas que entraña la programación del juego. Estos programas, sin embargo, no tienen posibilidades de derrotar a los maestros del juego.

Dentro de la tabla interior

El *backgammon* se juega sobre un tablero compuesto por 24 "puntas" triangulares en las que se mueven las piezas, en función del lanzamiento de un par de dados. El objetivo del juego es desplazar todas las piezas propias alrededor del tablero y colocarlas en el final antes que el oponente. Si los números de los dados son diferentes, uno puede mover una pieza la cantidad total de puntos que indican los dados, o bien mover dos piezas por separado según el número individual de cada dado. Cuando se mueve una pieza de acuerdo a la suma de los dados, se considera que está efectuando dos movimientos separados y se dice que "captura" en la punta fronteriza entre los dos movimientos. Las puntas ocupadas por dos o más piezas de un color no se pueden mover ni "capturar" por el lado oponente. Si una pieza ocupa una punta y es "capturada" por una pieza oponente, se la elimina del tablero y su dueño la debe volver a introducir antes de poder efectuar otro movimiento. La ilustración muestra un movimiento de la partida entre el programa de *backgammon* de Hans Berliner y el campeón del mundo, Luigi Villa, que se celebró en Montecarlo en 1980

Las blancas se mueve en sentido horario. Una vez que hayan entrado en la "table interior" de las blancas (puntas de la 19 a la 24), se las puede ir eliminando progresivamente. Las rojas juegan de forma similar, moviéndose en sentido antihorario, siguiendo las puntas por orden descendente

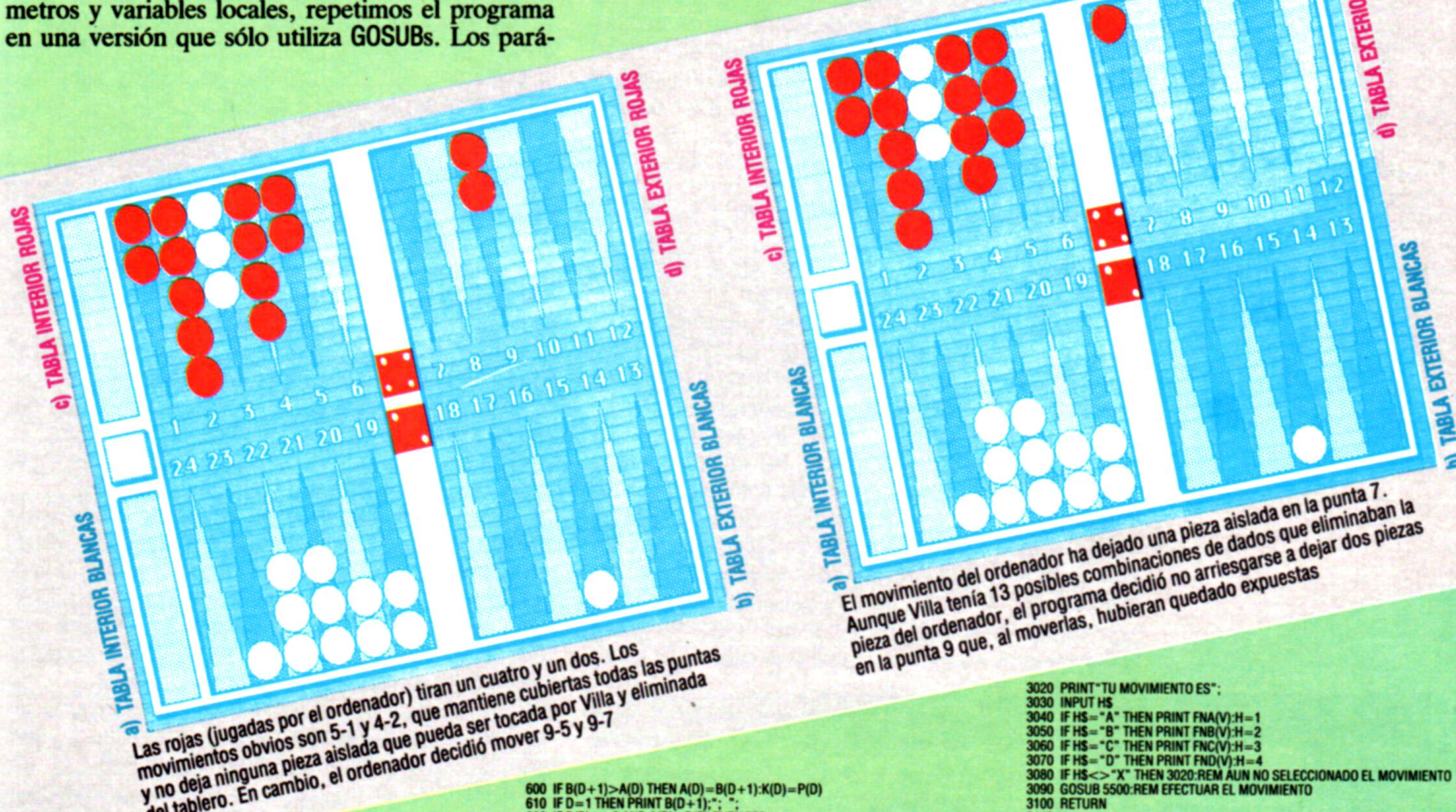


Para ilustrar los importantes conceptos de la búsqueda arborescente, ideamos un juego artificial que era casi una búsqueda pura. Este juego lo presentamos en el capítulo anterior en versión de BASIC BBC, haciendo un uso intensivo de las funciones recursivas con parámetros.

El juego permite que el jugador y el ordenador se turnen para seleccionar una de cuatro funciones que modifican un valor existente para producir uno nuevo. El jugador intenta reducir el valor a −255, mientras que la máquina intenta hacer que exceda de 255. En cada etapa el ordenador utiliza la minimaximización *alfa-beta* para seleccionar cuál de las cuatro funciones definidas es mejor para sus fines.

Teniendo presentes a los usuarios del Commodore 64 y el Spectrum, cuyos BASIC carecen de parámetros y variables locales, repetimos el programa en una versión que sólo utiliza GOSUBs. Los parámetros y las variables locales (a excepción de D, el contador de profundidad) se han sustituido por matrices, DIMensionadas en la línea 1100. Ahora D actuará como una especie de puntero de pila, llevando el registro de los elementos de las matrices a los que se esté accediendo en cada momento.

Estas dos rutinas se diferencian fundamentalmente en que todo ha de ir subindexado por D, en particular la matriz A(), utilizada para retener el mejor valor hallado hasta ahora, y B(), empleada para retener el peor. Ello asegura que los valores utilizados de *alfa* y *beta* se aplicarán a los niveles correctos del árbol.



Las rojas (jugadas por el ordenador) tiran un cuatro y un dos. Los movimientos obvios son 5-1 y 4-2, que mantiene cubiertas todas las puntas y no deja ninguna pieza aislada que pueda ser tocada por Villa y eliminada del tablero. En cambio, el ordenador decidió mover 9-5 y 9-7

El movimiento del ordenador ha dejado una pieza aislada en la punta 7. Aunque Villa tenía 13 posibles combinaciones de dados que eliminaban la pieza del ordenador, el programa decidió no arriesgarse a dejar dos piezas en la punta 9 que, al moverlas, hubieran quedado expuestas

El juego de los números

```
80 GOSUB 1000:REM INICIALIZACION
90 GOSUB 1600:REM INSTRUCCIONES
100 :
110 REM **** BUCLE PRINCIPAL DEL PROGRAMA ****
120 GOSUB 2000:REM PREPARAR NUEVO JUEGO
130 INPUT"QUIEN JUEGA PRIMERO (1=TU,2=YO)";H1
140 IF H1<1 OR H1>2 THEN 130
150 REM *** BUCLE DEL JUEGO ***
160 IF H1=1 THEN GOSUB 3000
170 REM ** TURNO DE LA PERSONA **
180 GOSUB 3500:REM VISUALIZACION DEL TABLERO
190 H1=1:REM SIEMPRE 1 TRAS EL 1ER CICLO
200 GOSUB 4000: REM COMPROBAR GANADOR
210 IF EG=0 THEN GOSUB 5000
220 REM ** TURNO DEL ORDENADOR **
230 GOSUB 3500:REM VISUALIZAR ESTADO DEL JUEGO
240 GOSUB 4000: REM COMPROBAR SI FINAL DEL JUEGO
250 IF EG=0 AND M<=33 THEN 150:REM BUCLE ATRAS
260 REM **** FINAL ****
270 GOSUB 6000: REM FELICITACIONES
280 INPUT"OTRA PARTIDA (1=SI,2=NO)";Y
290 IF Y<1 OR Y>2 THEN 280
300 IF Y=1 THEN 110: REM NUEVA PARTIDA
310 PRINT:PRINT "HASTA OTRA Y GRACIAS POR JUGAR"
320 END
330 :
500 REM **** MAXIMIZAR ****
510 D=D+1:C1=C1+1
520 IF D>=MD OR ABS(V(D))>HI THEN A(D)=V(D):D=D-1:RETURN
530 REM ** SI NO PROFUNDIZAR MAS **
540 P(D)=0
550 REM ** A TRAVES DEL ARBOL **
560 P(D)=P(D)+1:H=P(D):V=V(D):GOSUB 5500:REM REALIZAR MOVIMIENTO
570 IF D=1 THEN PRINT CHR$(64+H);"=";
580 D1=D+1
590 A(D1)=A(D):B(D1)=B(D):V(D1)=V:GOSUB 700:REM LLAMAR MINIMIZAR
600 IF B(D+1)>A(D) THEN A(D)=B(D+1):K(D)=P(D)
610 IF D=1 THEN PRINT B(D+1);";  ";
620 IF P(D)<=3 AND A(D)<B(D) THEN 550
630 IF D=1 THEN BV=A(D):HH=K(D):REM GUARDAR MEJOR HASTA AHORA
640 D=D-1:RETURN
650 :
700 REM **** MINIMIZAR ****
710 D=D+1:C2=C2+1
720 IF D>=MD OR ABS (V(D))>HI THEN B(D)=V(D):D=D-1:RETURN
730 (P(D)=0
740 REM ** A TRAVES DEL ARBOL **
750 P(D)=P(D)+1:H=P(D):V=V(D):GOSUB 5500:REM EFECTUAR MOVIMIENTO
760 D1=D+1:A(D1)=A(D):B(D1)=B(D):V(D1)=V
770 GOSUB 500:REM LLAMAR MAXIMIZAR
780 IF A(D+1)<B(D) THEN B(D)=A(D+1)
790 IF P(D)<=3 AND B(D)>A(D) THEN 740
800 D=D-1:RETURN
810 :
1000 REM **** INICIALIZAR ****
1010 BL$= " "
1020 REM ** DEFINICION DE LAS CUATRO FUNCIONES **
1030 DEF FNA(X)=2*X-7
1040 DEF FNB(X)=INT(X/2)+1
1050 DEF FNC(X)=4*X+17
1060 DEF FND(X)=3*X-4
1070 LO=-255:HI=255
1080 REM ** MATRICES QUE UTILIZA MINIMAX **
1090 D=16
1100 DIM V(D),A(D),B(D),P(D),K(D)
1110 RETURN
1120 :
1600 REM **** INSTRUCCIONES **
1610 PRINT"BIENVENIDO AL JUEGO DE LOS NUMEROS"
1620 PRINT"YO INTENTARE MAXIMIZAR, TU TAREA"
1630 PRINT"ES MINIMIZAR"
1640 PRINT"PARA VER EL EFECTO DE UN MOVIMIENTO PULSA:"
1650 PRINT"A, B, C 0 D. PULSA X PARA EFECTUARLO."
1660 PRINT:RETURN
1670 :
2000 REM **** PREPARACION ****
2010 M=0:V=INT(RND(1)*15)-8:REM ESTADO INICIAL
2020 EG=0
2030 PRINT"ESTADO INICIAL=";V
2040 RETURN
2050 :
3000 REM **** MOVIMIENTO DE LA PERSONA ****
3010 M=M+1:PRINT
3020 PRINT"TU MOVIMIENTO ES";
3030 INPUT H$
3040 IF H$="A" THEN PRINT FNA(V):H=1
3050 IF H$="B" THEN PRINT FNB(V):H=2
3060 IF H$="C" THEN PRINT FNC(V):H=3
3070 IF H$="D" THEN PRINT FND(V):H=4
3080 IF H$<>"X" THEN 3020:REM AUN NO SELECCIONADO EL MOVIMIENTO
3090 GOSUB 5500:REM EFECTUAR EL MOVIMIENTO
3100 RETURN
3110 :
3500 REM **** VISUALIZACION TABLERO ****
3510 PRINT:PRINT"          MOVIMIENTO";M;"—>";
3520 IF M<1 THEN RETURN
3530 PRINT CHR$(64+H);
3540 PRINT"=";V:PRINT:RETURN
3550 :
4000 REM **** PRUEBA GANADOR ****
4010 IF M<1 THEN RETURN
4020 EG=0
4030 IF V<LO THEN EG=-1
4040 IF V>HI THEN EG=1
4050 RETURN
4060 :
5000 REM **** MOVIMIENTO DEL ORDENADOR ****
5010 W=V:REM GUARDAR ESTADO ACTUAL
5020 M=M+1
5030 MD=6:REM MAX PROFUNDIDAD
5040 IF M<4 THEN MD=4
5050 IF M>8 THEN MD=8
5060 GOSUB 5200:REM —>H
5070 V=W:REM RESTAURAR ESTADO
5080 GOSUB 5500:REM EFECTUAR MOVIMIENTO
5090 RETURN
5100 :
5200 REM **** SELECCION MOVIMIENTO ****
5210 BV=LO:D=0
5220 V(1)=V:A(1)=LO:B(1)=HI
5230 GOSUB 500:REM MAXIMIZAR
5240 H=HH
5250 PRINT:INPUT"PULSA RETURN PARA CONTINUAR";Q
5260 RETURN
5270 :
5500 REM **** EFECTUAR UN MOVIMIENTO ****
5510 IF H=1 THEN V=FNA(V):RETURN
5520 IF H=2 THEN V=FNB(V):RETURN
5530 IF H=3 THEN V=FNC(V):RETURN
5540 IF H=4 THEN V=FND(V):RETURN
5550 :
6000 REM **** FELICITACIONES ****
6010 PRINT:PRINT"JUEGO TERMINADO"
6020 IF EG>0 THEN PRINT"HE GANADO YO"
6030 IF EG<0 THEN PRINT"HAS GANADO TU"
6040 IF EG=0 THEN PRINT"HA SIDO EMPATE"
6050 RETURN
```

Instrucciones transitorias

El CP/M cuenta con una gama de instrucciones que se cargan desde disco

Las instrucciones "transitorias" permiten que el usuario se comunique con las unidades de disco, administre archivos y controle otros dispositivos periféricos. Tras el encendido, el ordenador ejecutará una serie de comprobaciones para asegurarse de que estén presentes todos los componentes del sistema y que éstos sean correctos. Asimismo, el procesador abrirá canales a todos los periféricos que estén conectados. El ordenador enviará códigos a los diversos periféricos, en parte para comprobar que estén funcionando y, en parte, para prepararlos para recibir datos. Todo este proceso se conoce como *inicialización*.

Al inicializar algunas interfaces de periféricos, el ordenador espera que el periférico le devuelva un mensaje antes de proseguir con cualquier otra acción, tal como activar la ROM de BASIC. Un ejemplo de esto se puede apreciar cuando se instala un cartucho de juegos que, en efecto, se hace cargo del sistema operativo del ordenador y efectúa toda las entradas y salidas con el propio procesador, sin remitirse a la ROM de BASIC. De modo similar, cuando un ordenador compatible con CP/M abre un canal para su unidad de disco, espera a que la unidad le devuelva un mensaje.

El resultado de la inicialización del ordenador sobre la unidad de disco es que se obliga al cabezal de lectura/escritura del dispositivo a leer la primera pista del disco de sistema CP/M. Si no hay ningún disco presente, la unidad continuará girando hasta que se inserte un disco. La pista cero contiene el programa *cargador básico (boodstrap loader)*, que proporciona al ordenador las instrucciones necesarias para la carga en memoria del resto del programa CP/M. Si en la pista cero no hay ningún programa cargador, el ordenador generará un error DOS.

Extensión	Explicación	Ejemplo
ASM	Se requiere para arch. fuente de leng. máq.	CODEPROG.ASM
BAK	Copia de un arch. de texto creado mediante el editor	MEMO.BAK
BAS	Indica un archivo fuente en BASIC	PROG.BAS
COM	Extensión necesaria para un arch. trans. o de instrucciones	PIP.COM
HEX	Asignado a un arch. hexadecimal a nivel máq.	GRAPHIC.HEX
INT	Otorgado a un programa en BASIC compilado	GAMES.INT
PRN	Necesaria para producir listados de progs. en leng. assembly	CODEPROG.PRN
SUB	Arch. utilizado para ejecutar instrs. en lotes *(batch)*	SPOOLER.SUB
$$$	Archivo temporal creado mediante el editor	PHONE.$$$

Después de que se haya terminado de cargar el CP/M en el ordenador, aparecerá en la pantalla un cursor intermitente junto a un símbolo A> (aunque algunas máquinas utilizan OA>). Se dice que éste es un *aviso* e indica que el sistema está listo para recibir una instrucción. También informa de cuál es la unidad de disco CP/M que está en ese momento en uso y en la que espera leer y escribir información. Se dice que la unidad a la cual está accediendo el CP/M en el momento es la *unidad actualmente conectada (logged)*. En este caso, dado que apenas acabamos de cargar CP/M, todavía estamos en la unidad A.

Esto está muy bien si sólo poseemos una unidad de disco o bien si deseamos usar una sola. No obstante, si contamos con dos unidades de disco y queremos utilizar ambas, hemos de indicar al CP/M que busque la unidad B. Esto se consigue mediante la instrucción B y pulsando RETURN. El CP/M comprobará entonces que esté presente la unidad B y si en la misma hay en este momento un disco. De ser así, aparecerá en la pantalla el aviso B>, indicando que estamos ahora en la unidad B. Pero no estamos limitados a dos unidades. El CP/M puede acceder hasta a cuatro, aludiéndose a las otras como C y D, respectivamente.

Si bien no es muy usual que una misma máquina tenga instaladas cuatro unidades de disco separadas, muchos ordenadores, como el 380Z y el 480Z de Research Machines, poseen unidades de doble cara. En tales casos, lo normal es llamar A y B a las caras superiores de las unidades y C y D a las inferiores.

Tras conectar con el CP/M, lo normal es averiguar qué archivos están disponibles. Para hacer esto, debemos examinar el directorio del disco entrando la instrucción DIR o dir (el CP/M no diferencia entre mayúsculas o minúsculas). Ello hará que se visualice una lista de archivos, incluyendo todas las instrucciones que se pueden cargar y ejecutar desde CP/M.

Quizá parezca que tener que cargar y ejecutar un "archivo de instrucciones" con el mero fin dc, por ejemplo, obtener información de estado sobre un archivo retenido en disco, es un proceso innecesariamente largo. ¿Por qué no podrían las instrucciones cargarse en RAM tras el encendido, y ejecutarse directamente cuando ello fuera necesario? La razón principal de este método, aparte de la obvia de ahorrar espacio de memoria, es facilitar la compatibilidad entre sistemas diferentes.

Como puede imaginar, el CP/M contiene instrucciones que nos permiten manipular archivos retenidos en disco. Si bien las aplicaciones CP/M han de ser estándares con el objeto de que sean portables de una máquina a otra, en el transcurso de los años muchos fabricantes han adaptado el sistema de acuerdo a las necesidades de sus propios ordenadores. Por ejemplo, en CP/M no hay ninguna instrucción de formateo de disco estándar, de modo que cada fabricante le añade a la lista de instrucciones transitorias del CP/M esta función en particular, por lo que muchos sistemas trabajan de forma diferente.

Como vimos en el capítulo anterior, el nombre de un archivo CP/M se compone de un nombre primario, un punto y una extensión. Esta última afecta a la forma en que se carga el archivo en el ordenador. En los discos de sistema CP/M hay un cierto número de archivos de instrucciones que contienen las instrucciones transito-

rias del CP/M, llevando todas ellas la extensión .COM. Ello significa que cuando se carguen en el ordenador se ejecutarán automáticamente.

Al objeto de cargar un archivo de instrucciones, usted simplemente digita el nombre primario, pulsa RETURN y se ejecutará el programa de instrucciones. Otros archivos con diferentes extensiones se cargan también de forma diferente, e iremos viéndolos con posterioridad.

Anteriormente hemos examinado el uso de la instrucción DIRectorio. En CP/M hay una instrucción transitoria asociada llamada STAT, que proporciona información adicional sobre el disco y los archivos en él retenidos. Al ejecutarse, STAT visualizará la cantidad de memoria libre disponible para nuevos archivos. La instrucción visualizará, asimismo, cierta información relativa al disco, como cuánta memoria queda todavía disponible para otros archivos y qué clase de facilidades de lectura/escritura se permiten. Por ejemplo, el mensaje R/W significa que se puede leer el disco y escribir en él, mientras que R/O significa que el disco sólo se puede leer o, mejor dicho, que está protegido contra la escritura.

Además, la instrucción STAT también puede "bloquear" un disco de modo que sólo pueda ser leído. Esto se consigue digitando STAT D:=R/O, donde D alude al carácter DRIVE. Todo intento ulterior de escribir en este disco concreto provocará un mensaje BDOS. La instrucción también visualizará las extensiones de nombre de archivo y la cantidad de sectores de archivo lógicos utilizados por cada registro, y mediante el empleo de STAT es posible examinar el tamaño de archivos individuales simplemente digitando STAT seguido del nombre del archivo. Asimismo, STAT se puede usar para examinar y, si así se requiere, modificar el estado de los periféricos que estén instalados. A modo de ejemplo, la instrucción STAT DEV visualizará una lista de todos los dispositivos de entrada/salida (incluyendo la pantalla) que estén conectados.

Una de las características más importantes de un sistema operativo de disco es la capacidad de transferir archivos de un disco a otro. En CP/M, esta operación la lleva a cabo PIP (*Peripheral Interchange Program*: programa de intercambio de periféricos). Esta instrucción no hace otra cosa que copiar de disco a disco; asimismo, permite la salida de archivos por impresora o por otros dispositivos de entrada/salida.

Para poder utilizar PIP se debe cargar y ejecutar la instrucción digitando PIP y pulsando RETURN. Observe que ahora el aviso pasa a ser un asterisco, indicando que se está ejecutando PIP y que se está esperando la instrucción siguiente. El formato para la utilización de PIP es: D:COPYNAME = D:SOURCENAME. A modo de ejemplo, vamos a suponer que tenemos un archivo de texto HCAC.TXT en un disco y queremos copiarlo en un segundo disco bajo el nombre WORK.TXT. Debemos sacar el disco de sistema de la unidad y colocar el disco que contenga HCAC.TXT en la unidad A, y colocar el segundo disco en la unidad B. De modo que para copiar HCAC.TXT de la unidad A a la unidad B, el formato es: B:WORK.TXT=A:HCAC.TXT.

Observe que cuando se utiliza PIP la unidad objeto se coloca primero en la instrucción, antes de la unidad fuente, que va a continuación, y que cuando se alude a archivos que no son instrucciones debe incluirse el nombre completo, incluyendo la extensión. Tras haber completado esta operación, usted puede constatar si el archivo se ha copiado correctamente examinando el directorio de la unidad B.

Aunque tenga una sola unidad de disco también puede utilizar PIP, pero las unidades fuente y de destino serán, por supuesto, ambas A. El CP/M copiará sectores del archivo en la memoria y le informará cuándo se deben cambiar los discos de modo que se puedan volcar los sectores desde la memoria al disco de destino.

En caso de que deseáramos enviar HCAC.TXT a la impresora, utilizaríamos una forma similar de PIP. En este caso, impartimos la instrucción: PIP LPT:=B:HCAC.TXT. Hay varios puntos a destacar en esta instrucción, el primero de los cuales es el hecho de que PIP no se cargará separadamente del resto de la instrucción. Esto es posible en CP/M puesto que usted no tiene necesariamente que esperar a que PIP se cargue antes de que pueda ocurrir alguna otra cosa.

En segundo lugar, el dispositivo de destino no es un canal de unidad sino una impresora, que es la razón por la cual utilizamos la instrucción LPT (*Line PrinTer*). En esta instrucción, hemos incluido asimismo la letra de unidad que indica dónde hallar el archivo HCAC.TXT. Si la unidad B estuviera conectada en ese momento, esta inclusión no sería necesaria. En realidad, bajo algunas versiones de CP/M no es necesaria ni aunque estemos en la unidad A. Esto se debe a que el CP/M buscará en todas las unidades de disco disponibles antes de generar un error de "archivo no hallado".

Al transferir un archivo de un disco a otro también cambiamos el nombre del archivo. Ésta no es una exigencia de PIP, dado que un archivo se puede copiar y conservar el mismo nombre. Sin embargo, supongamos que deseáramos cambiar el nombre de un archivo en un disco sin tener que copiarlo. En este caso, utilizaríamos la instrucción REN (de *rename*), de modo que si deseáramos cambiar el nombre de WORK.TXT por HCAC.TXT, emplearíamos la instrucción REN HCAC.TXT=WORK.TXT, recordando que siempre se coloca primero el nombre objeto. Ello creará un nuevo nombre de archivo en el directorio y borrará el anterior.

Del mismo modo, cuando deseamos borrar un archivo de un disco, digitamos ERA seguido por el nombre del archivo. Como con todas las instrucciones destructivas de esta clase, ésta debe ser usada con sumo cuidado. El CP/M debe saber a qué disco se está refiriendo usted, de lo contrario quizá acabe borrando una versión equivocada del archivo. De modo que, aunque no es necesario, es un buen hábito incluir siempre el nombre de la unidad de la instrucción; por ejemplo, ERA B:HCAC.TXT.

El ASM

El CP/M, tal como se implementa en la mayoría de las máquinas, incluye su propia utilidad de ensamblador denominada ASM. Aunque con frecuencia los programadores lo ignoran, se trata de un potente ensamblador con capacidad en ensamblaje condicional, así como todas las directivas de ensamblador estándares. El único inconveniente de la utilización del ASM es que originalmente se desarrolló para usar con el ensamblador 8080, que luego se reemplazaría por el Z80. Si bien el código objeto generado por el 8080 se ejecutará en el Z80, usted encontrará que un ensamblador 8080 tal como el ASM no siempre aceptará los mnemotécnicos estándares del Z80. En el libro *Programming the Z80*, de Rodnay Zaks, hallará una relación completa de las diferencias. No obstante, si pretende utilizar regularmente el ASM en una máquina Z80, le aconsejamos que se consulte con Digital Research, que puede proporcionarle un juego de macrorrutinas para ejecutar con ASM

Trazado de memoria

Finalizada la construcción del trazador digital, nos concentraremos en su calibrado y en el desarrollo del software para controlarlo

El primer paso para desarrollar software para el trazador consiste en calibrar el hardware. La ejecución del programa *Calibrado* que ofrecemos aquí nos proporciona cuatro valores digitales de potenciómetro que corresponden a cuatro posiciones críticas del brazo: posición cero del brazo 1, posición cero del brazo 2, posición en 90º del brazo 1 y posición en 90º del brazo 2. Estos valores deben entrarse en el programa del trazador digital en las líneas 1200, 1210, 1240 y 1250, como parte del procedimiento definir—parametros. Los otros dos parámetros que pueden variar de un trazador a otro son las longitudes de los brazos, y éstas deben entrarse en el programa en las líneas 1220 y 1260. Observe que las longitudes de los brazos deben estar expresadas en milímetros y que el brazo 2 incluye la distancia desde el extremo del brazo hasta el cruce de los ejes.

Habiendo obtenido estos valores de calibrado, podemos utilizarlos para convertir en ángulos las lecturas digitales proporcionadas por los potenciómetros (a través del convertidor de analógico a digital del propio BBC Micro). Esta información angular se puede convertir en desplazamientos horizontales y verticales de la mira desde el punto de montaje del brazo. En general, un ángulo se puede calcular aplicando la siguiente fórmula:

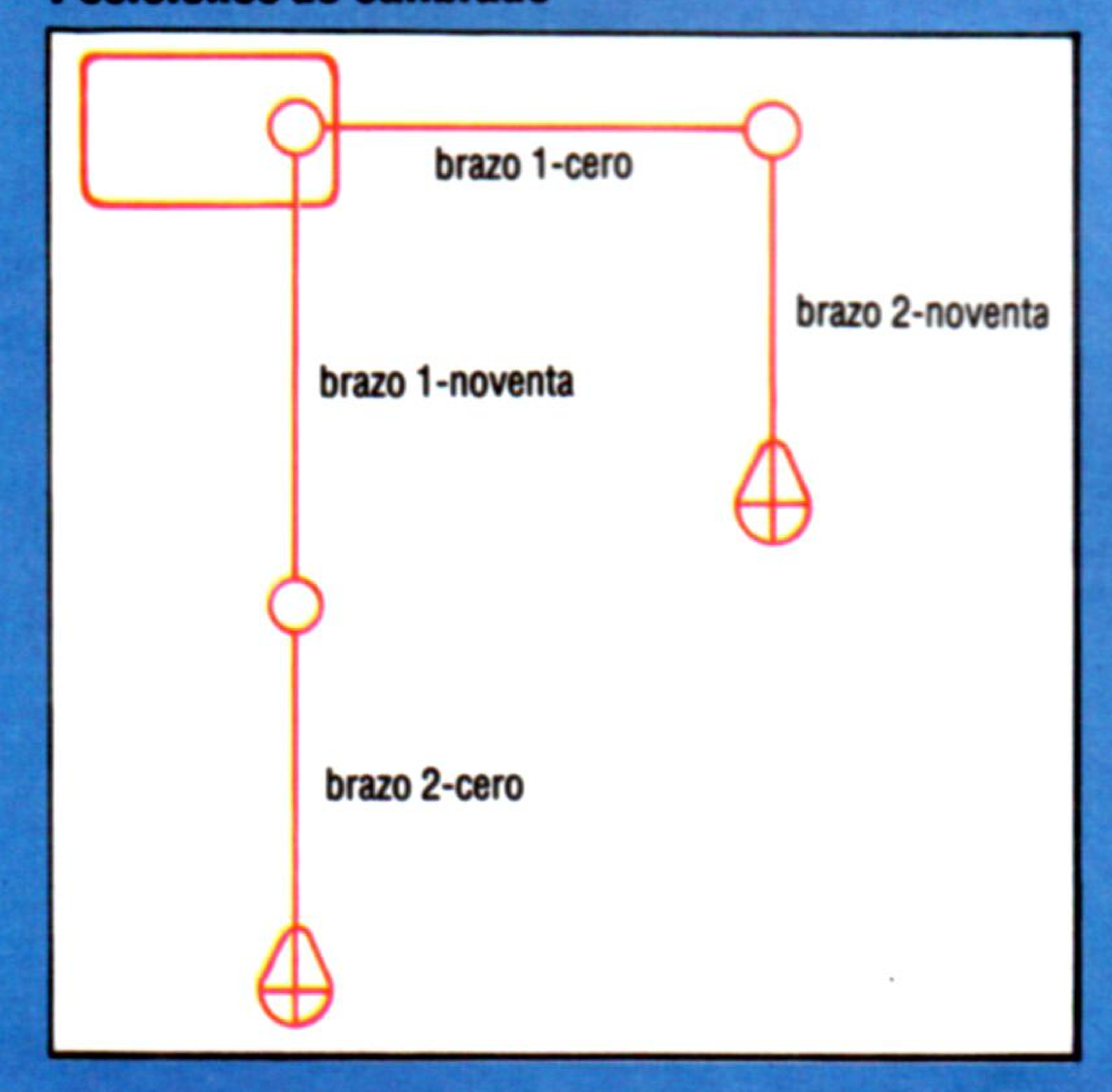
```
angulo=(ADVAL(n)—brazocero)*90/(brazonoventa-brazocero)
```

Puesto que la segunda parte de esta expresión es una constante, es mejor calcular el valor al comien-

Calibrado

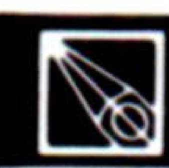
Coloque el tablero del trazador hacia arriba de modo que la caja plástica quede en el rincón superior izquierdo. Para poder calibrar con total éxito el trazador necesitamos 4 informaciones: la posición en 0º del brazo 1, la posición en 90º del brazo 1, la posición en 0º del brazo 2 y la posición en 90º del brazo 2. Mientras se ejecuta el programa de calibrado, desplace el brazo del trazador de modo que el brazo 1 quede paralelo respecto al margen superior del tablero y el brazo 2, perpendicular. Apunte en un papel los valores digitales del brazo 1 y el brazo 2. Mueva el ensamblado del brazo de modo que ahora los brazos 1 y 2 estén en línea y paralelos al borde izquierdo del tablero. Vuelva a anotar las lecturas para los brazos 1 y 2. Pulse una tecla del teclado para pasar a la modalidad de ángulo. Entre las 4 lecturas que ha apuntado en respuesta a las preguntas y después compruebe que los ángulos visualizados para ambos brazos sean correctos. De no ser así, repita este proceso hasta que lo sean. Conserve la nota donde apuntó los 4 valores digitales del brazo, porque serán necesarios posteriormente.

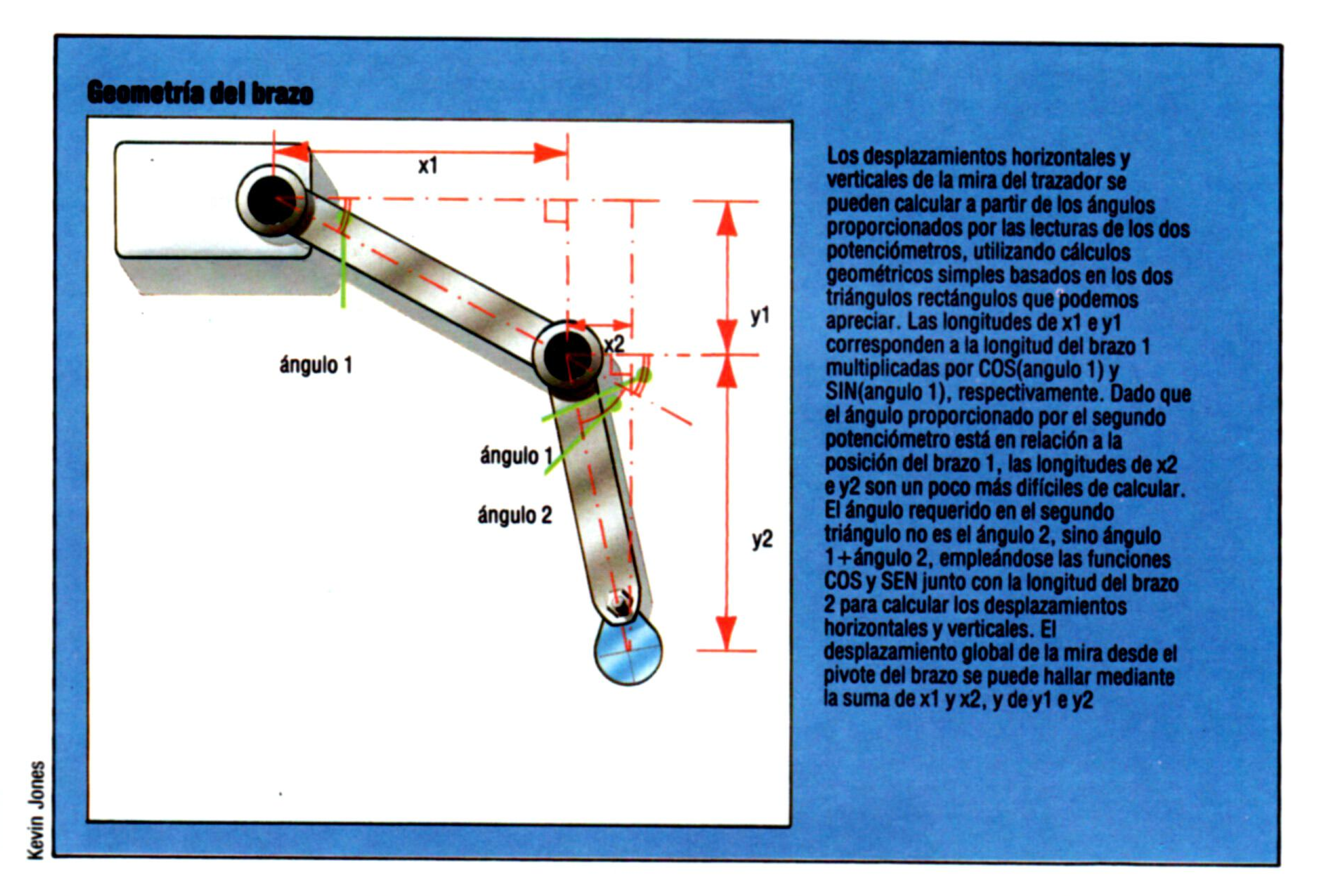
Posiciones de calibrado




```
1000 REM **** CALIBRADO TRAZADOR BBC ****
1010 PROCcalibrar
1020 PROCvisualizar_angulo
1030 END
1040 DEF PROCcalibrar
1050 CLS
1060 PRINT TAB(5,10);"ANGULO 1";TAB(20);"ANGULO 2"
1070 REPEAT
1080 PRINT TAB(5,12);SPC(40)
1090 PRINT TAB(5,12);ADVAL(1);TAB(20);ADVAL(2)
1100 PROCdemora(600)
1110 A$=INKEY$(1)
1120 UNTIL A$<>""
1130 ENDPROC
1150 DEF PROCvisualizar_angulo
1160 PROCdefinir_angulos
1170 CLS
1180 PRINT TAB(5,10);"ANGULO 1";TAB(20);"ANGULO 2"
1190 REPEAT
1200 PROCcalc_angulos
1210 PRINT TAB(5,12);SPC(40)
1220 PRINT TAB(5,12);angulo1;TAB(20);angulo2
1230 PROCdemora(1000)
1240 A$=INKEY$(1)
1250 UNTIL A$<>""
1260 ENDPROC
1280 DEF PROCdefinir_angulos
1290 CLS
1300 PRINT TAB(5,10);:INPUT"Posicion cero 1er. brazo";uno_cero
1310 PRINT TAB(5);:INPUT"Posicion noventa 1er. brazo";uno_noventa
1320 PRINT TAB(5);:INPUT"Posicion cero 2do. brazo";dos_cero
1330 PRINT TAB(5);:INPUT"Posicion noventa 2do. brazo";dos_noventa
1340 uno_factor=90/(uno_noventa-uno_cero)
1350 dos_factor=90/(dos_noventa-dos_cero)
1360 ENDPROC
1380 DEF PROCcalc_angulos
1390 angulo1=INT((ADVAL(1)-uno_cero)*uno_factor)
1400 angulo2=INT((ADVAL(2)-dos_cero)*dos_factor)
1420 DEF PROCdemora(demora)
1430 LOCAL I
1440 FOR I=1 TO demora:NEXT I
1450 ENDPROC
1460 PROCdemora(1000)
```

Kevin Jones





Los desplazamientos horizontales y verticales de la mira del trazador se pueden calcular a partir de los ángulos proporcionados por las lecturas de los dos potenciómetros, utilizando cálculos geométricos simples basados en los dos triángulos rectángulos que podemos apreciar. Las longitudes de x1 e y1 corresponden a la longitud del brazo 1 multiplicadas por COS(angulo 1) y SIN(angulo 1), respectivamente. Dado que el ángulo proporcionado por el segundo potenciómetro está en relación a la posición del brazo 1, las longitudes de x2 e y2 son un poco más difíciles de calcular. El ángulo requerido en el segundo triángulo no es el ángulo 2, sino ángulo 1+ángulo 2, empleándose las funciones COS y SEN junto con la longitud del brazo 2 para calcular los desplazamientos horizontales y verticales. El desplazamiento global de la mira desde el pivote del brazo se puede hallar mediante la suma de x1 y x2, y de y1 e y2

Kevin Jones

zo del programa y utilizarlo en todos los posteriores cálculos de ángulo. Si calculamos:

```
factor=90*/(brazonoventa-brazocero)
```

podemos volver a expresar la fórmula como:

```
angulo=(ADVAL(n)-brazocero)*factor
```

El procedimiento calc_xy calcula los ángulos actuales tomados por los dos potenciómetros y los utiliza para calcular los desplazamientos x e y de la mira. La geometría involucrada es bastante directa. Con respecto al diagrama de geometría del brazo, x1 e y1 se relacionan con la longitud del brazo 1 mediante las funciones SIN y COS:

```
x1=COS(angulo1)*longitud_brazo1
y1=SIN(angulo1)*longitud_brazo1
```

El cálculo de x2 e y2 es un poco más complicado, dado que angulo2 está en cero cuando el brazo 2 se halla en línea con el brazo 1. El ángulo en la esquina del triángulo rectángulo es, por consiguiente, angulo1+angulo2 y:

```
x2=COS(angulo1+angulo2)*longitud_brazo2
y2=SIN(angulo1+angulo2)*longitud_brazo2
```

Los desplazamientos x e y totales se pueden hallar sumando, respectivamente, x1 y x2, e y1 e y2.

Las fórmulas utilizadas en el procedimiento cal_xy son ligeramente diferentes de las reseñadas anteriormente. Las variables de longitud del brazo se reemplazan por variables escaladas. Estas variables están relacionadas con las longitudes de los dos brazos, pero también incorporan un factor de modo que los desplazamientos x e y se realicen a escala para que quepan en el sistema de coordenadas de gráficos del BBC Micro.

La forma más obvia de guardar visualizaciones creadas en la pantalla es realizar en código máquina una salvaguardia del área de RAM utilizada por MODE 1. Siempre y cuando no se haya desplazado la pantalla desde el último CLS, esta zona estará entre &3000 y &7FFF. Sin embargo, el empleo de la instrucción *SAVE tiene el inconveniente de que no se le pueden pasar nombres de archivo utilizando una variable en serie. Por ejemplo, la instrucción:

```
*SAVE archivo$ 3000 8000
```

guardará el archivo con el nombre archivo$ en vez del nombre en serie retenido en la variable archivo$. Esto significa que el usuario no puede guardar fácilmente visualizaciones en pantalla en disco o cinta bajo diferentes nombres de archivo. Por fortuna, existe una forma de evitar este problema, empleando la llamada al sistema operativo OSCLI. Esta instrucción se puede utilizar para ejecutar un bloque de códigos ASCII retenido en la memoria, como si los caracteres correspondientes se hubieran entrado directamente desde el teclado. Para usar esta llamada debemos pasar en los registros X e Y las direcciones de comienzo del bloque ASCII en forma *lo-byte/hi-byte*. Para llevar a cabo *SAVE y *LOAD, por lo tanto, ensamblamos la instrucción (junto con el nombre de archivo proporcionado por el usuario) en una serie y colocamos (POKE) los valores ASCII de los caracteres que componen la serie en un bloque reservado de memoria. Tras establecer los registros X e Y de modo que apunten al comienzo de este bloque, la instrucción allí almacenada se lleva a cabo llamando a OSCLI en la dirección &FFF7. Ésta es la función que cumple el procedimiento instruccion_oscli de la línea 2170.

Además de guardar y cargar pantallas, la pantalla se puede limpiar digitando C, y regresar al menú principal digitando M. La segunda selección de este menú no se puede efectuar en esta etapa, puesto que constituirá la segunda parte del programa y la ofreceremos en el próximo capítulo. Se pueden seleccionar nuevos colores de primer plano en cualquier momento pulsando 1, 2 o 3. Estos colores son

los de primer plano por defecto de MODE 1, y se los puede cambiar mediante el empleo de la instrucción VDU 19. El programa que ofrecemos se ejecutará correctamente en modalidad de mano alzada.

Programa del trazador

La primera mitad del programa *Trazador digital*, que ofrecemos aquí, permite utilizar el trazador a mano alzada. La segunda mitad incluye rutinas que permitirán emplearlo para producir puntos, líneas y curvas individuales. En modalidad de mano alzada, hay seis facilidades disponibles. El botón a presión, montado en el trazador, actúa como un mando de "lápiz arriba/lápiz abajo", permitiendo que el usuario pase de uno a otro. El botón está cableado en la puerta analógica como el botón de disparo de una palanca de mando y se puede detectar desde software examinando los dos bits inferiores del valor devuelto en ADVAL(0). Cada uno de los dos bits estará establecido en uno si se pulsa el botón de disparo correspondiente. Por tanto, (ADVAL(0)AND3) <> 0 indica que se ha producido la pulsación de un botón de disparo y se puede emprender la acción adecuada. En este caso, una variable denominada flagart oscilará entre 0 y 1 a cada sucesiva pulsación del botón. El valor de flagart se comprueba en el procedimiento dibujar y se toma la decisión ya sea de DRAW o bien de MOVE a una nueva posición (según cual sea el valor).

Se utiliza una pequeña cruz como cursor para indicar la posición actual de la mira del trazador en la pantalla. La misma se puede borrar y desplazar sin alterar los datos de fondo mediante el uso de la modalidad de trazado Exclusive-OR. Todas las líneas dibujadas en esta modalidad (seleccionada mediante GCOL3) se pueden borrar volviendo a dibujarlas exactamente en la misma posición. Por lo tanto, el procedimiento dibujar calcula las nuevas coordenadas del curso y después llama a un procedimiento para dibujar el cursor, que borrará el cursor viejo volviéndolo a dibujar en modalidad Exclusive-OR. El cursor de gráficos (invisible para nosotros) se desplaza (MOVE) hasta la antigua posición del cursor y se dibuja una línea o se realiza un desplazamiento hasta las nuevas coordenadas, según cual sea el valor de flagart. Por último, los valores actuales de las coordenadas se almacenan en antx y anty, como preparación para la próxima ocasión en que se llame al procedimiento dibujar.

```
   6 REM ** TRAZADOR DIGITAL BBC **
1080 PROCdefinir_parametros
1090 REPEAT
1100 MODE 1
1110 REM APAGAR CURSOR
1120 VDU 23,1,0;0;0;0;
1130 PROCmenu
1140 CLS
1150 IF resp$="1" THEN PROCmanoalzada ELSE PROCelastico
1160 UNTIL flagfinal=1
1170 END
```

Se proporcionan las longs. de los brazos y las lecturas de calibrado de los pots.

```
1190 DEF PROCdefinir_parametros
1200 uno_cero=14820
1210 uno_noventa=45020
1220 uno_longitud=250
1230 uno_factor=90/(uno_noventa-uno_cero)
1240 dos_cero=450
1250 dos_noventa=25300
1260 dos_longitud=222+28
1270 dos_factor=90/(dos_noventa-dos_cero)
1280 REM ** FACTORES DE CONVERSION DE MM A COORDENADAS
     GRAFICOS **
1290 escala=1023/460
1300 uno_escala=escala* uno_longitud
1310 dos_escala=escala* dos_longitud
1320 REM EXPLORAR SOLO 2 CANALES ADC
1330 *FX 16,2
1340 flagfinal=0:flagart=0:color=1
1350 DIM save% 30:DIM x(3),y(3)
1360 ENDPROC
1380 DEF PROCmanoalzada
1390 PROC free_inform
1400 REPEAT
1410 IF(ADVAL(0) DIV 256)<>0 THEN PROCdibujar
1420 IF(ADVAL(0)AND 3)<>0 THEN PROCarticular_lapiz
1430 resp$=INKEY$(1):IF resp$<> "" THEN PROCpulsacionlibre
1440 UNTIL flagsalida=1
1450 ENDPROC
1470 DEF PROCdibujar
1480 PROCcalc_xy
1490 PROCcursor(antx,anty)
1500 MOVE antx,anty
1510 IF flagart=0 THEN DRAW x,y ELSE MOVE x,y
1520 PROCcursor(x,y)
1530 antx=x:anty=y
1540 ENDPROC
1560 DEF PROCcursor(cx,cy)
1570 GCOL 3,3:REM MODALIDAD TRAZADO EOR
1580 MOVE cx,cy-16
1590 PLOT 1,0,32
1600 PLOT 0,-16,-16
1610 PLOT 1,32,0
1620 GCOL 0,color
1630 ENDPROC
```

El sig. proc. lleva a cabo los cálculos para convertir las lecturas de los pots. en comps. verticales y horizontales x e y

```
1650 DEF PROCcalc_xy
1660 angulo1=RAD((ADVAL(1)-uno_cero)*uno_factor)
1670 angulo2=RAD((ADVAL(2)-dos_cero)*dos_factor)
1680 x=uno_escala*COS(angulo1)+dos_
     escala*COS(angulo1+angulo2)
1690 y=1023-(uno_escala*SIN(angulo1)+dos_
     escala*SIN(angulo1+angulo2))
1700 ENDPROC
1720 DEF PROCmenu
1730 flagsalida=0:flagfinal=0
1740 PRINT TAB(5,10);"Por favor, elija"
1750 PRINT TAB(5);"1...Modalidad mano alzada"
1760 PRINT TAB(5);"2...Modalidad elastica"
1770 PRINT:PRINT TAB(5);"Pulse 1 o 2"
1780 REPEAT:resp$=GET$:UNTIL resp$="1" OR resp$="2"
1790 ENDPROC
1810 DEF PROCarticular_lapiz
1820 flagart=1-flagart
1830 REPEAT UNTIL (ADVAL(0) AND 3)=0
1840 ENDPROC
1860 DEF PROCpulsacionlibre
1870 IF resp$="C" THEN CLS:PROCfree_inform:ENDPROC
1880 IF resp$="M" THEN flagsalida=1
1890 IF resp$"S" THEN PROCguardar_pantalla:PROCfree_
     inform:ENDPROC
1900 IF resp$="L" THEN PROCcargar_pantalla:PROCfree_
     inform:ENDPROC
1910 PROCcambio_color
1920 ENDPROC
1940 DEF PROCfree_inform
1950 PROCcal_xy:antx=x:anty=y:PROCcursor(antx,anty)
1960 PRINT TAB(1,1);SPC(79)
1970 PRINT TAB(1,1);"S=Save L=Load M=Menu C=Limpiar"
1980 GCOL 0,1:MOVE 0,920:DRAW 1280,920
1990 ENDPROC
2010 DEF PROCcambio_color
2020 resp=VAL(resp$)
2030 IF resp<1 OR resp>3 THEN ENDPROC
2040 color=resp
2050 ENDPROC
```

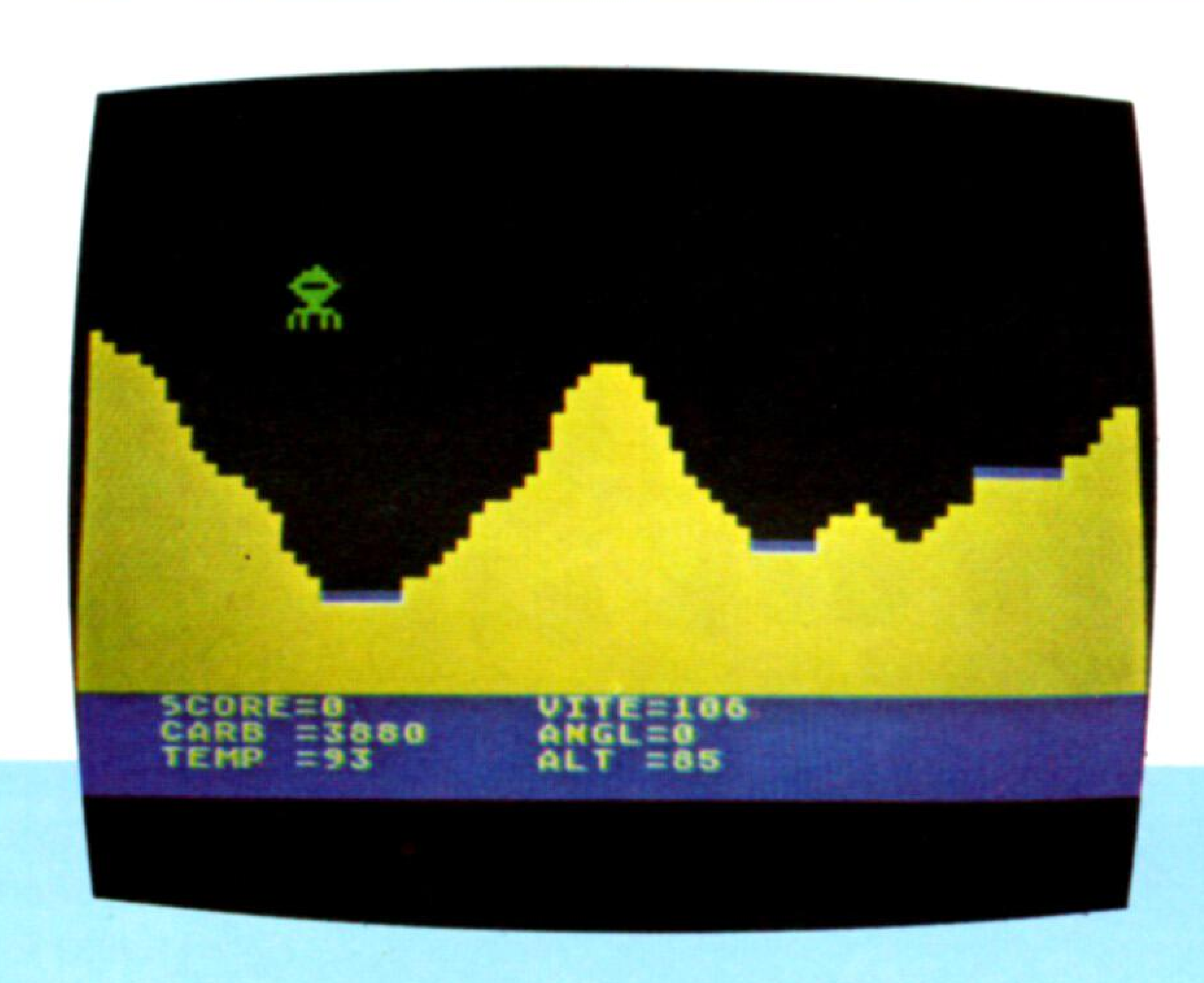
Estos procs. permiten guardar o cargar el área de pant. como área RAM usando *SAVE y *LOAD indirectamente a través de OSCLI

```
2070 DEF PROCguardar_pantalla
2080 PROCcursor(x,y):REM CURSOR APAGADO
2090 REPEAT
2100 PRINT TAB (1,1);SPC(79)
2110 INPUT TAB (1,1);"GUARDAR
     NOMBREARCHIVO";archivo$:archivo$=archivo$+".S"
2120 UNTIL LEN(archivo$)<8
2130 archivo$="*SAVE"+archivo$+"3000 8000"
2140 PROCinstruccion_oscli(archivo$)
2150 ENDPROC
2170 DEF PROCinstruccion_oscli(a$)
2180 FOR I=0 TO LEN(a$)-1
2190 save%?I=ASC(MID$(a$,I+1,1))
2200 NEXT I
2210 save%?I=13:REM AÑADIR CR
2220 X%save% MOD 256:Y%=save% DIV 256
2230 CALL &FFF7:REM LLAMAR OSCLI
2240 ENDPROC
2260 DEF PROCcargar_pantalla
2270 REPEAT
2280 PRINT TAB(1,1);SPC(79)
2290 INPUT TAB(1,1);"LOAD
     NOMBREARCHIVO";archivo$:archivo$=archivo$+".S"
2300 UNTIL LEN(archivo$)<8
2310 archivo$="*LOAD"+archivo$
2320 PROCinstruccion_oscli(archivo$)
2330 ENDPROC
```



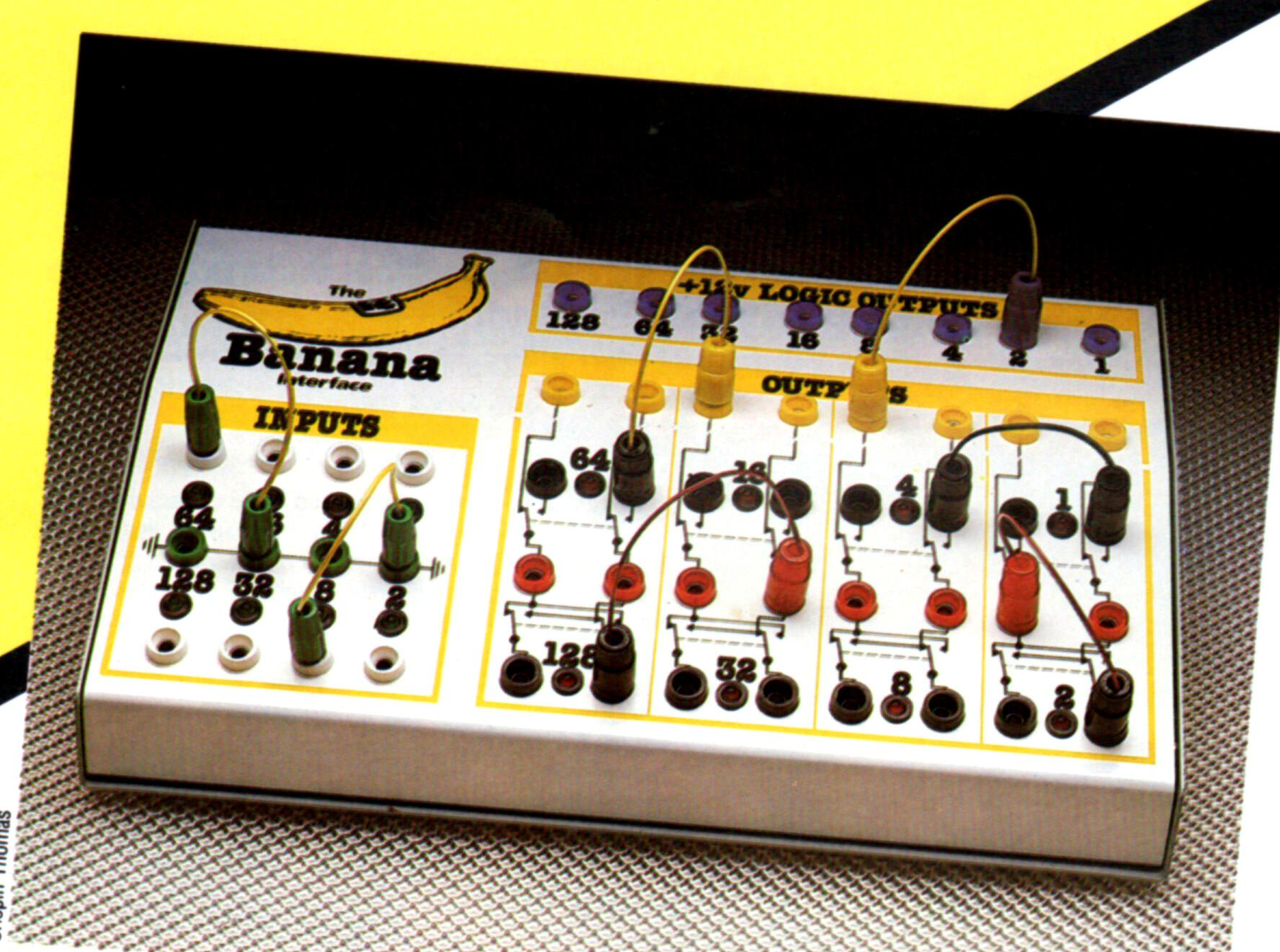
En la Luna

Este juego ha sido escrito por Paul Dunning para el microordenador Atari. Para posar su nave espacial en la superficie lunar es preciso utilizar la palanca de mando.



Su misión consiste en posar su nave espacial en una de las tres pistas azules que se visualizan en la pantalla. Si la velocidad es excesiva usted se estrellará; una velocidad de 5 parece adecuada. Su puntuación se calcula en función de la velocidad a la cual aluniza y de la pista que haya escogido. Si tiene éxito en su alunizaje, vuelve a partir con un suplemento de 500 litros de combustible. El juego finaliza cuando haya agotado su stock de carburante.

```
0 REM * ALUNIZAJE DE PAUL DUNNING *
1 DATA 108, 60, 81, 40, 53, 40, 60, 40, 64, 40, 72, 40, 40, 40, 53,
  40, 60, 40, 64, 40, 72, 40, 40, 40, 53, 40, 60, 40, 64, 40, 60, 40,
  72, 60
2 DATA 108, 60, 96, 80, 60, 40, 64, 40, 72, 40, 81, 80, 72, 40, 64,
  40, 72, 60, 96, 40, 85, 40, 53, 80, 40, 40, 45, 40, 50, 40, 53, 40,
  60, 40
3 DATA 68, 40, 72, 40, 81, 40, 53, 80, -1
4 CLR :DIM A$(100), H$(11), E$(60):H$ = " a Lari":POKE 752,
  1:RE=500
5 AA=1:CA=4000: TE=100:SC=0:AN=0:VI=AN:POKE 53278,
  1:GOSUB 900:GOSUB 4000
6 DATA 11, 12, 13, 13, 14, 15, 17, 19, 21, 22, 23, 23, 24
7 DATA 25, 26, 29, 30, 31, 33, 33, 33, 33, 33, 33, 31
8 DATA 31, 30, 29, 28, 26, 25, 25, 24, 23, 21, 18, 16, 15
9 DATA 14, 14, 14, 15, 17, 19, 21, 23, 24, 25, 26, 27, 29
10 DATA 29, 29, 29, 29, 27, 26, 26, 25, 26, 27, 28, 28
11 DATA 27, 26, 25, 25, 23, 23, 23, 23, 23, 23, 23, 21, 21
12 DATA 20, 18, 17, 17
29 FOR I=1 TO 8:POKE 53247+I,0:NEXT I:GOTO 495
30 END
120 REM POKE 53248, 30
200 REM ***** JUEGO JUGADOR *****
210 POKE 559,0:POKE 710, 148:POKE 712,0
215 POKE 752,1:AL=110
230 RESTORE 235:FOR S2=1 TO 60:READ S3:E$(S2,S2)=CHR$
    (S3):NEXT S2
235 DATA 104, 104, 104, 74, 72, 176, 11, 160, 2, 177, 203, 136,
    145, 203, 200, 200, 208, 247, 104, 74, 72
236 DATA 176, 11, 160, 253, 177, 203, 200, 145, 203, 136
237 DATA 136, 208, 247, 104, 74, 72, 176, 7, 198, 205
238 DATA 165, 205, 141, 0, 208, 104, 74, 176, 7, 230, 205, 165,
    205, 141, 0, 208, 96, 0, 0
240 A=PEEK(106)-16:POKE 54279,A:POKE204, A+4:POKE 203,
    0:PMBASE=A*256
250 POKE 53277, 3:POKE 704,40:POKE 53248, 150:POKE 205,
    RND(1)*130+70:POKE 53256, 0
260 FOR I=PMBASE+1024 TO PMBASE+1280:POKE I, 0:NEXT I
270 RESTORE 290
280 FOR I=PMBASE+1155 TO PMBASE+1209:READ B:POKE I,
    B:NEXT I
290 DATA 8, 8, 60, 60, 126, 126, 195, 195, 126, 126, 60, 60, 24, 24,
    126, 126, 165, 165
300 DATA 165, 165, 165, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
310 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
320 POKE 559,62:GOTO 390
370 ST=STICK(0)
380 CO=PEEK(53252):IF CO>0 THEN GOSUB 500
385 POKE 53278,1
390 IF AA=1 THEN FOR Q=0 TO 100:A=USR(ADR(E$), 14):SOUND
    0,255, 8, 15:NEXT Q:AA=0:A=USR(ADR(E$), 11):SOUND
    0,0,0,0
400 POKE 53278,3
410 IF ST=14 AND CA>=30 THEN SOUND 0,255, 8, 15:GOTO 420
415 SOUND 0,0,0,0
420 IF ST=7 OR ST=11 AND CA>=5 THEN SOUND 1,100,8,3:GOTO
    430
425 SOUND 1,0,0,0
430 IF CA<=0 THEN CA=0:GOTO 455
431 IF ST=7 THEN AN=AN+0.05:CA=CA-5
432 IF ST=11 THEN AN=AN-0.05:CA=CA-5
440 HP=HP+AN: IF HP>1 THEN HP=0:SR=7
450 IF HP<-1 THEN HP=0:SR=11
455 VI=VI+0.02:VP=VP+VI:IF VP>1 AND SR=1 5 THEN
    VP=0:SR=13:AL=AL-1
456 IF VP>1 AND SR=11 THEN VP=0:SR=9:AL=AL-1
457 IF VP>1 AND SR=7 THEN VP=0:SR=5:AL=A-L-1
458 IF VP<-1 THEN VP=0:SR=14:AL=AL+1
460 IF ST=14 AND CA>29 THEN VI=VI-0.05:CA=CA-30
480 A=USR(ADR(E$),SR):SR=15
485 ? " PUNTUACION="; SC; ";CHR$(127);" VEL.=" ;VI*100;" "
486 ? " CARB =";CA;" ";CHR$(127);" ANG =";AN*100;"
487 ? "TIEMPO =";INT(TE);" ";CHR$(127);"ALT =";AL;" "
488 ? CHR$(28);CHR$(28);CHR$(28);
489 TE=TE-0.1
490 IF TE=0 THEN 720
492 GOTO 370
495 GOTO 210
500 IF CO=4 AND VI>0.5 THEN 600
505 IF CO=4 AND VI>0.4 THEN 700
510 IF CO=2 THEN 700
520 RETURN
600 FOR Q=1 TO 2:FOR W=255 TO 100 STEP -1
610 SOUND 0,W,10,15:SOUND 1,W-10,10,15:SOUND 2,W-20,10,15
620 NEXT W:NEXT Q
630 SS=0,5-VI:SO=SC:IF SS=0 THEN SC=SC+5 0:GOTO 650
640 IF AL<5 THEN SC=SC+(SS*100)*10:GOTO 650
642 IF AL<20 THEN SC=SC+(SS*100)*20:GOTO 650
644 IF AL<60 THEN SC=SC+(SS*100)*5
650 IF CA<2500 THEN CA=CA+(SC-SO*2)
655 FOR Q=0 TO 3:SOUND Q,0,0,0:NEXT Q:AA=1:POKE 53278,
    1:GOSUB 800:GOTO 210
700 FOR Q=0 TO 50:POKE 712, RND(1)*255:SOUND 0,255, 8,
    15:SOUND 1,240,8,15:SOUND 2,220,8,15:NEXT 0
705 SOUND 0,0,0,0:SOUND 1,0,0,0:SOUND 2,0,0,0
706 IF CA<30 THEN CA=0:GOTO 720
710 POKE 712,0:AA=1:GOSUB 800:GOTO 210
720 POKE 712,0:GOTO 810
800 VP=0:HP=0:AN=0:VI=0:TE=100:RETURN
810 IF SC>RE THEN GOSUB 1010
820 GRAPHICS 17:SETCOLOR 4,1,0
830 ? #6;"   FINALIZADO"
835 ? #6:? #6;"      RECORD ";RE
836 ? #6:? #6;"      ";H$
840 ? #6:? #6;"      PULSE START "
850 POKE 53248,0
860 IF PEEK(53279)=6 THEN RESTORE :GOTO 5
870 GOTO 860
899 END
900 REM INTRO
910 GRAPHICS 17:SETCOLOR 4,1,0:E=0
920 ? #6;"     ALUNIZAJE "
930 ? #6:? #6;" DE P. DUNNING SEP.82"
940 ? #6:? #6;"PALANCA DE MANDO (0)"
950 ? #6:? #6;"        ALTO     "
960 ? #6:? #6;"
970 ? #6;"  IZQUIERDA<      >DERECHA"
980 ? #6
985 POKE 53248,0
990 READ M:IF M<0 THEN 998
992 READ P
995 SOUND 0,M,10,15:FOR PP=0 TO P:NEXT PP
997 GOTO 990
998 ? #6;"     buena suerte"
999 FOR X=1 TO 13:FOR Y=0 TO 255 STEP 20:SOUND
    0,Y,10,15:NEXT Y:NEXT X:SOUND 0,0,0,0:RETURN
1000 RESTORE 1:GOTO 990
1010 H$=" ":TE=30:GRAPHICS 18:SETCOLOR 4,1,0:? #6;"      NO
     ESTA MAL   ":? #6:X=1:Y=1:RE=SC
1011 ? #6;"  a b c d e f g h "
1012 ? #6
1013 ? #6;"  i j k l m n o p "
1014 ? #6
1015 ? #6;"  q r s t u v w x "
1016 ? #6
1017 ? #6;"  y z . -     R E "
1018 POKE 53248,0
1020 FOR Q=10 TO 245:SOUND 0,Q-10,8,15:SOUND
     1,Q,8,15:SOUND 2,Q+10,8,15:NEXT Q
1030 POSITION X,Y:? #6;"XXX": POSITION X,Y+1:? #6;"X": POSI-
     TION X+2,Y+1:? #6;"X": POSITION X,Y+2:? #6;"XXX"
1040 IF STICK(0)<>7 THEN 1050
1045 POSITION X,Y:? #6;"    ": POSITION X,Y+1:? #6;"  ": PO-
     SITION X+2,Y+1:? #6;" ":POSITION X,Y+2:? #6;"    "
1046 IF X>13 AND Y=7 THEN X=1:Y=1:GOTO 1050
1047 IF X>13 THEN X=1:Y=Y+2:GOTO 1050
1048 X=X+2
1050 IF STICK(0)<> 11 THEN 1060
1055 POSITION X,Y:? #6;"":POSITION X,Y+1:? #6;"":POSITION
     X+2,Y+1:? #6;"":POSITION X,Y+2:? #6;"    "
1056 IF X<3 AND Y=1 THEN X=15:Y=7:GOTO 1060
1057 IF X<3 THEN X=15:Y=Y-2:GOTO 1060
1058 X=X-2
1060 POSITION 6,10:? #6; "TIEMPO ";INT(TE);" ":TE=TE-0.1:IF
     TE=0 THEN 1100
1065 FOR PP=0 TO 20:NEXT PP
1069 IF STRIG(0)<>0 THEN 1080
1071 LOCATE X+1,Y+1, XX:IF XX = ASC("E") THEN 1100
1072 IF XX=ASC("R") AND LEN(H$)>1 THEN H$ =
     H$(1,LEN(H$)-1>:GOTO 1080
1073 IF LEN(H$)>10 THEN 1080
1075 H$(LEN(H$)+1)=CHR$(XX)
1080 POSITION 4,11:? #6,H$;" "
1099 GOTO 1030
1100 FOR Q=0 TO 3:SOUND Q,0,0,0:NEXT Q
1110 RETURN
4000 GRAPHICS 5
4010 RESTORE 6:COLOR 2
4020 FOR X=0 TO 79
4030 READ Y
4040 PLOT X,39:DRAWTO X,Y
4050 NEXT X
4060 COLOR 3
4070 PLOT 18,32:DRAWTO 23,32
4080 PLOT 50,28:DRAWTO 54,28
4090 PLOT 67,22:DRAWTO 73,22
4100 RETURN
```



Crispin Thomas

Banana sensible
La Banana Interface, así llamada por los enchufes banana que utiliza, permite la conexión en interface de dispositivos eléctricos analógicos con un ordenador (que utiliza señales digitales). De esta forma se pueden construir sistemas de realimentación que permitan que el ordenador controle acontecimientos del mundo exterior y responda a ellos

Bananarama

Bajo su piel
La Banana Interface se puede utilizar para una gran variedad de aplicaciones de control, tanto en el hogar como en el laboratorio. Por ejemplo, el entorno de la habitación que vemos en la ilustración se puede poner casi por completo bajo el control de un ordenador conectado a una Banana Interface. En el cuarto se pueden instalar dispositivos de percepción, tales como termostatos o sensores luminosos. La información recibida desde estos dispositivos las puede analizar el ordenador y compararla con rangos óptimos preprogramados. Cualquier ajuste del entorno que resulte necesario (como elevar la temperatura, bajar las persianas o encender las luces por la tarde) se puede llevar a cabo a través de las líneas de salida de la interface

SC Sensor de contacto
CC Control de calefacción
IL Interruptor para reducir la luz
ME Motor eléctrico
AA Altavoz de alarma
DH Detector de humo
TM Termostato
CF Célula fotosensible



La Banana Interface constituye un interesante aporte en el campo de los dispositivos controlados por ordenador

Anteriormente ya hemos examinado numerosos dispositivos que permiten que los ordenadores controlen maquinarias, y en nuestro apartado *Bricolaje* también hemos ilustrado cómo se pueden controlar numerosos dispositivos. Uno de los últimos que han salido al mercado, dirigido básicamente a las escuelas, es la Banana Interface, diseñada para permitir controlar varios proyectos diferentes desde el BBC Micro o el Commodore 64.

Esencialmente el concepto que subyace en la Banana Interface es idéntico al de la caja buffer que construimos en el apartado *Bricolaje*, si bien el Banana es algo más sofisticado. Manipulando los patrones de bits binarios que contienen los registros de datos y de dirección de datos del ordenador, se pueden transmitir señales de 5 V a dispositivos externos. Además, debido a que las puertas para el usuario y la impresora son bidireccionales, la máquina puede recibir señales desde el periférico que le permiten "saber" algo sobre el estado del dispositivo (su posición, p. ej.) o si se ha activado algún interruptor disparador. Por consiguiente, es posible que el ordenador pueda, mediante una combinación de señales de entrada y salida, guiar con precisión a un robot y reaccionar ante las circunstancias cambiantes del entorno del mismo.

La interface se conecta al ordenador a través de un trozo de cable plano que se enchufa en las puertas para la impresora y para el usuario. El otro extremo del cable está instalado en un conector de 40 vías de la propia interface. Para poder operar la Banana Interface se requieren dos fuentes de alimentación eléctrica CD, la primera de las cuales es una corriente de 5 V que se suministra desde el ordenador a través del cable plano. Esta corriente se utiliza para ejecutar la lógica de la interface a lo largo de las líneas de datos.

La segunda fuente de alimentación se proporciona externamente mediante un par de conectores *minijack* situados en la parte posterior del dispositivo junto a la interface del cable plano. Lamentablemente, el fabricante, Castle Associates, ha decidido no proporcionar con la interface una fuente de 12 V. La razón de ello es que el Banana está destinado básicamente para ser usado en las escuelas, las cuales suelen contar con amplios suministros de

la corriente adecuada en sus laboratorios y talleres. Por consiguiente, dado que las escuelas no necesitan una fuente de alimentación externa, la empresa ha optado por excluirla y mantener reducido el precio del dispositivo.

Esto significa que los particulares interesados en adquirir un Banana se encuentran ante la perspectiva de tener que conseguir su propia fuente de alimentación. Puesto que las fuentes de 12 V con pequeños amperajes no se consiguen con facilidad, probablemente se tendrá que operar la interface mediante unas cuantas pilas conectadas en serie (la empresa recomienda un amperaje de 600 mA, aunque la máquina que utilizamos operó sin dificultades con una fuente de 10 V y un amperio).

Cuando la Banana Interface se hallaba en fase de planificación, Castle Associates consultó a profesores de diseño y tecnología. Esta consulta repercutió positivamente en las especificaciones del dispositivo, en particular en cuanto concierne a su uso en las escuelas. Con la carcasa completamente construida en acero, la interface es uno de los dispositivos más robustos que existen para cualquier ordenador personal.

Esta preocupación por la duración también se ha aplicado a los componentes internos de la interface. La placa de circuito impreso se ha soldado a los conectores hembra, manteniendo firmemente en su sitio el delicado tablero. El resultado es un periférico que probablemente pueda soportar sin problemas las manipulaciones erróneas y que, para ser objeto de un daño serio, requerirá la intervención de un escolar muy decidido a lograrlo.

En la parte superior de la carcasa hay varias hileras de conectores *minijack* divididos en tres grupos. A la izquierda de la interface hay ocho líneas de entrada blancas, con cuatro líneas a tierra, verdes, colocadas entremedio. Cada enchufe hembra está numerado 1, 2, 4, 8, y así sucesivamente, que corresponden al número retenido en cada una de las posiciones de bits del registro. Junto a las posiciones de bits hay LEDs que se utilizan para dar información respecto a cuál de los bits está *high* (indicando que a través de esa línea de datos concreta está pasando una corriente de 5 V).

El ordenador puede registrar los cambios en las líneas de entrada, y la forma de demostrarlo es estableciendo *high* todas las líneas. Hacer un PEEK del registro producirá un valor de 255, pero si conectáramos un cable desde una de las líneas de tierra a uno de los conectores de entrada blancos, el voltaje caería a cero y el valor del registro caería consecuentemente, según cuál de las líneas se utilizara. Por lo tanto, se podrían construir aplicaciones simples de interrupción de circuitos, como alarmas antirrobo.

Las líneas de salida

A la derecha de las entradas, y ocupando la mayor parte de la superficie, están las líneas de salida. Al igual que las de entrada, hay ocho posiciones básicas (numeradas 1, 2, 4, y así hasta 128), LEDs correspondientes a cada una de las posiciones de bits y cuatro conectores *minijack*. Mediante circuitos de relé se conectan pares de conectores hembra. La colocación (POKE) de un número en el registro en la dirección &FE61 disparará los relés.

Conectando los motores eléctricos y fuentes de alimentación adecuadas a través de los terminales, estos relés se pueden utilizar para activar los motores y posicionarlos como se desee. Empleándola de esta forma, desde la Banana Interface se pueden controlar hasta cuatro motores eléctricos a la vez. Obviamente, la forma más eficaz de controlar los motores eléctricos (que se podrían conectar a una tortuga para el suelo o a un brazo-robot) es activarlos a través de los interruptores de salida y monitorizar sus movimientos a través de las puertas de entrada.

A lo largo de la parte superior de la interface hay otro grupo de ocho puertas lógicas de alta velocidad, cada una de las cuales produce una corriente de 12 V. Estas líneas de salida se utilizan para activar desde la interface motores paso a paso pudiéndose activar hasta siete de ellos al mismo tiempo.

Parece que la Banana Interface puede tener un buen mercado en las escuelas y los centros de investigación como medio auxiliar para la enseñanza del control básico por ordenador de dispositivos eléctricos. Es claro que existen numerosas aplicaciones para las cuales se podría utilizar en demostraciones y experimentos. También podría tener amplias aplicaciones en pequeñas empresas que deseen automatizar algunos de sus procesos de producción.

Desde el punto de vista del aficionado, la utilidad de un dispositivo de estas características ya no está tan clara. Ciertamente se lo puede emplear en casa para numerosos experimentos realizados como pasatiempo y es probable que le enseñe muchísimas cosas sobre las técnicas de control por ordenador. En este sentido, el amante serio de la robótica bien puede considerar la adquisición de uno. No obstante, a los usuarios del BBC Micro y del Commodore 64 cuyo interés por estos dispositivos sea sólo superficial, la Banana Interface les resultará excesivamente cara por sus limitaciones.

Kevin Jones

BANANA INTERFACE

DIMENSIONES
300×200×63 mm

INTERFACES
Conector en paralelo de 40 vías desde las puertas para la impresora y el usuario del BBC Micro y el Commodore 64. Puertas de salida, ocho conectores de salida de 12 V

VENTAJAS
La Banana Interface está bien construida y permite una amplia variedad de aplicaciones de control por ordenador

DESVENTAJAS
La falta de una fuente de alimentación de 12 V adecuada limita su mercado potencial ya sea a los aficionados serios o bien a los establecimientos educativos o de investigación

Interés declarado

Examinaremos la naturaleza declarativa del PROLOG y la forma como maneja proposiciones.

El PROLOG está diseñado para dar al programador un medio de describir la estructura lógica de un problema utilizando una base de datos de hechos y reglas *if-then*. Esta enunciación "declarativa" del problema la emplean los mecanismos deductivos del PROLOG para producir respuestas a los interrogantes que uno ha planteado. La tarea del programador se transforma, por consiguiente, de tener que decirle al ordenador, con todo detalle, cómo usar las operaciones de un lenguaje (para resolver un conjunto predefinido de problemas) a tener que proporcionar una enunciación clara y lógica del conocimiento requerido para resolver los problemas, dejando luego que sea el PROLOG el que lleve a cabo la tarea.

La sintaxis del PROLOG es muy sencilla, pero la terminología es bastante oscura. La construcción más importante es el *término*. Un término puede ser una *constante* (como "árbol", "juan" o "25"), una *variable* o una *estructura*. Las estructuras se crean a partir de constantes y variables. Una estructura muy común en los programas en PROLOG es el *hecho*. Éste se compone de un predicado (que debe ser una constante) ya sea solo, o bien seguido de una lista de argumentos (que pueden ser constantes o variables) entre paréntesis:

```
predicado1.
predicado2(argumento1,argumento2,
argumento3).
```

Como vimos en el primer capítulo de esta serie, se puede considerar al predicado como una relación existente entre sus argumentos. En la frase en castellano "Venusianos comen árboles", "comen" es el predicado que relaciona a venusianos con árboles. En PROLOG, podríamos escribir este hecho como:

```
come(venusianos,árboles).
```

Los términos como éste se pueden acumular para formar grandes grupos de hechos de forma muy similar a como se acumulan registros en archivos de bases de datos. En el diagrama tenemos una base de datos de hechos acerca de quién come qué. Sin ningún código extra, esta base de datos ya nos ofrece un programa en PROLOG.

Para ejecutar el programa necesitamos establecer una proposición para que el intérprete la demuestre.

¿Quién come qué?

```
come(venusianos,árboles).
come(jovianos,rocas).
come(terrícolas,gachas).
come(mercurianos,gusanos).
come(neptunianos,gachas).
come(marcianos,marcianos).
```

En PROLOG, los "hechos" se expresan como un predicado (en este caso, "come"), en solitario o seguido por una lista de argumentos. Se los puede utilizar para proporcionar al programa una "base de datos de conocimientos" a la cual pueda interrogar luego el usuario

Supongamos que deseamos saber si los jovianos comen rocas. Planteamos el interrogante digitando:

```
?-come(jovianos,rocas).
```

tras el aviso del sistema (por lo general ?-). El PROLOG examina entonces nuestros hechos y, si encuentra una pareja para la proposición, dice yes, lo que significa "sí, a partir del programa se puede ver que come(jovianos,rocas) es verdadero". Si después preguntamos:

```
?-come(marcianos,gachas).
```

el PROLOG responderá no.

Podemos lograr que el PROLOG nos haga más cosas empleando variables en nuestra interrogación. La convención del PROLOG es que los nombres de variables empiezan con una letra en mayúscula, mientras que las constantes estándares empiezan con minúscula. Por lo tanto, si deseamos saber quiénes comen gusanos, podemos preguntar:

```
?-come(Criatura,gusanos).
```

a lo que el PROLOG responderá:

```
Criatura=mercurianos
```

y después hará una pausa, esperando una entrada. Ello significa que el PROLOG ha descubierto que, estableciendo la variable Criatura en el valor mercurianos, puede probar que la proposición es verdadera.

En este punto podemos ya sea entrar RETURN, para indicar que estamos satisfechos con la respuesta, o bien podemos digitar un punto y coma para indicarle al PROLOG que busque otra forma de demostrar la proposición. En este ejemplo, sin embargo, no hay ninguna otra forma, de modo que el PROLOG nos responderá no. No obstante, si le solicitamos que busque quién come gachas, habrá dos respuestas posibles: terrícolas y neptunianos. De modo que, en respuesta a:

```
?-come(Criatura,gachas).
```

el PROLOG dice:

```
Criatura=terrícolas
```

y después digitamos ;, a lo que el PROLOG responderá:

```
Criatura=neptunianos
```

Si hubiera más soluciones, la pulsación del punto y coma tras cada una obligaría al PROLOG a hallarlas todas. En el próximo capítulo veremos exactamente cómo funciona esto, pero veamos antes otros tres importantes conceptos: *reglas, cadenas de inferencia* y *retroceso*.

Uno o más términos constituyen una cláusula, que puede expresar una *regla*. Una cláusula siempre posee un encabezamiento compuesto por un término, y éste puede ir seguido por su cuerpo, formado por uno o más términos. El encabezamiento y el cuerpo se separan mediante el operador simbólico :-, que normalmente se lee como "si". Por consiguiente, tenemos:

```
término1:-término2,término3,término4.
```

Las comas entre los términos del cuerpo se pueden leer como un AND lógico, de modo que se podría interpretar que esta estructura de nuestro ejemplo significa que "el término1 es válido SI el término2 AND el término3 AND el término4 son todos válidos".

Para clarificar mejor esto, imaginemos que deseamos averiguar si alguna de las criaturas de nuestra base de datos es caníbal. Podemos escribir una cláusula que nos dé una definición de caníbal:

```
caníbal(Criatura):-
come(Criatura,Criatura).
```

Ésta es una cláusula con un término en su cuerpo y podemos leerla como una "regla" que afirma que: una criatura es caníbal si esa criatura come el mismo tipo de criatura. Si añadimos esta regla y después preguntamos:

```
?-caníbal(X).
```

el PROLOG empareja esta proposición con el encabezamiento de nuestra nueva cláusula. Dado que esta nueva cláusula



no es verdadera de forma automática, primero se debe demostrar que los términos del cuerpo de la cláusula son verdaderos. De modo que el PROLOG toma uno por uno, de izquierda a derecha, y los establece como proposiciones, tal como si se los hubiera digitado en forma de interrogaciones. En el cuerpo hay un solo término:

```
...,come(Criatura,Criatura).
```

y concuerda con el hecho (come(marcianos,marcianos). Esto a su vez establece a X=marcianos (X era el nombre de variable de la interrogación original) y el PROLOG responderá:

```
X=marcianos
```

En este caso, come(marcianos,marcianos) se encontró como un hecho en la base de datos. Sin embargo, si no hubiese sido un simple hecho sino otra regla, el PROLOG habría tenido que abocarse a otras subproposiciones para tratar de demostrar la veracidad del encabezamiento de la regla.

El programa en PROLOG del diagrama ilustra cómo podría suceder esto. Si planteamos el interrogante:

```
?-color_de(marty,Color).
```

queriendo averiguar el color de marty, el PROLOG primero toma nuestra interrogación como su proposición. Luego encuentra una regla para decidir si color_de(marty,rosa) es verdadero y encuentra que ello es así si (:-) el talante de marty es feliz. Estableciendo luego esto como su siguiente proposición, descubre una regla que afirma que marty es feliz si puede programar en PROLOG. De modo que puede_programar_en(marty,prolog) se convierte en la siguiente proposición.

Esto podría continuar de este modo a través de muchísimos niveles; pero, en este caso, se interrumpe aquí porque no hay ninguna regla ni ningún hecho que demuestre, o bien que indique cómo demostrar, que marty es un programador de PROLOG.

El PROLOG, simplemente, no se da por vencido en este punto. Lo que hace, sin embargo, es admitir su fracaso en la proposición actual y luego retornar a la proposición previa para ver si hay alguna forma alternativa de demostrarlo. Pero eso también falla (no se puede demostrar que el talante de marty sea feliz). De modo que el PROLOG *retrocede* nuevamente para ver si hay alguna forma alternativa de demostrar la primera proposición. De hecho, sí la hay. El PROLOG encuentra el hecho color_de(marciano,azul), establece marciano=marty y la proposición triunfa.

Este proceso de abrirse paso a través de largas cadenas de reglas y plantearse a sí mismo una nueva proposición que demostrar a cada paso, es la forma en la que el PROLOG resuelve todos sus interrogantes. El método se denomina *retroceso* porque, si un camino determinado a través de las reglas no produce una respuesta, el PROLOG recorre el camino hacia atrás hasta llegar a un punto de opción anterior, y luego prosigue en esa nueva dirección. De esta forma, hallará su respuesta o bien explorará cada uno de los caminos posibles a través de las reglas. Por este motivo los programas en PROLOG se comprenden mejor como *árboles*, o cadenas de proposiciones, en vez de como listas de enunciaciones.

¿Cuándo se pone azul un marciano?

```
color_de(Marciano,rosa):-talante_de
  (Marciano,feliz)
color_de(Marciano,azul)
talante_de(Marciano,feliz):-
puede_programar_en(Marciano,prolog).
talante_de(Marciano,triste).
puede_programar_en(marty,basic).
puede_programar_en(miranda,prolog)
```

En este conjunto de términos, observe que los nombres de las variables empiezan con mayúscula. El símbolo :- se puede interpretar como un *if* (si) lógico.

El programa Alvey

En la actualidad, el PROLOG desempeña un papel esencial en el programa de investigación Alvey para el desarrollo de "superordenadores" de la quinta generación. El programa se estableció en 1983 tras el informe del Alvey Committee, creado para aconsejar sobre las posibilidades de la investigación en colaboración como respuesta a la iniciativa tomada por los japoneses con su Proyecto de Ordenadores de la Quinta Generación. La cooperación es el factor primordial del proyecto Alvey: ha estado estrechamente vinculado con el proyecto ESPRIT (*European Fifth Generation*: quinta generación europea) y se han hecho contactos para establecer vínculos con los japoneses. Se han aprobado alrededor de 100 proyectos, todos los cuales responden a una estrategia de planificación central que se revisa y actualiza anualmente. Empleando las características del PROLOG de "base de datos inteligente", una máquina de este tipo se podría implementar directamente como un sistema experto. Otras aplicaciones del PROLOG que se están considerando actualmente incluyen su uso como parte de una interface para lenguaje natural. El PROLOG se presta particularmente para el proceso de lenguajes naturales y, dado que esta área es un aspecto esencial de las máquinas auténticamente "amables con el usuario" y fundamental para la investigación en materia de quinta generación, parece un hecho seguro que el PROLOG continuará ocupando un papel primordial en el desarrollo de nueva tecnología



Influencia paterna
Profesor Bob Kowalski, a quien se considera "padre de la programación lógica", cuyas ideas respecto a la programación y los sistemas lógicos utilizara A. Colmerauer para desarrollar en la Universidad de Marsella el primer intérprete de PROLOG

Ganar o perder

El diseño del algoritmo final de nuestro juego de simulación incluye la eventual participación en una revuelta local y los preparativos del viaje de regreso

Al entrar en la fase final del juego, ha concluido el intercambio comercial y se han cargado a bordo del barco las mercancías adquiridas al jefe de los nativos del Nuevo Mundo. A la mañana siguiente, la tripulación se reaprovisionará de agua fresca y alimentos e iniciará el viaje de regreso. Se han comerciado todas las mercancías, a excepción de las armas, y al regresar a puerto la venta de las especias, figurillas y perlas reportará algún beneficio.

Sin embargo, antes de emprender el viaje de regreso, se produce un nuevo acontecimiento. Durante la noche, un rival del jefe visita en secreto el barco. El insurgente desea comprarle sus armas con el fin de derrotar al jefe, y ofrece 30 perlas por cada una. Si usted acepta el trato obtendrá un inmenso beneficio, pero debe tener en cuenta algunos puntos:

Si el jefe descubre que usted ha vendido las armas, seguramente se vengará de alguna forma en caso de que la rebelión fracasase. No obstante, el margen de beneficios del viaje podría ser tan reducido que quizá usted considere que las ganancias extras bien valen el riesgo a correr. Puede, por supuesto, rechazar la oferta, en cuyo caso el jefe, como gesto de agradecimiento, le proporcionará provisiones gratuitamente para el viaje de regreso e incluso podría incluir 50 perlas como recompensa. Existe la posibilidad, si vende las armas, de que la revuelta triunfe; pero si fracasa, las consecuencias serán desastrosas. El jefe, tras enterarse de su doble juego, ordenará que se prenda fuego a su barco, con lo cual el juego terminará.

Si usted puede evitar este desastre final, el barco emprenderá el regreso a puerto; la tripulación se halla en buena forma, los vientos son favorables y el viaje se realizará en ocho semanas. Cuando el barco llegue a puerto, se pueden vender las mercancías, pagarle a la tripulación y, con suerte, obtener algún beneficio. Si no hubiera suficiente dinero para pagarle a la tripulación, se podría vender el barco para cubrir gastos. El informe final del programa calcula una hoja de balance para el viaje y visualiza información en el sentido de si ha terminado con pérdidas o ganancias.

El programa principal llama, en la línea 893, a una subrutina de la línea 10300, que se ocupa de la oferta hecha por el rival del jefe de adquirir sus armas. El programa comprueba primero si hay algún arma disponible para el intercambio, porque si no hubiera ninguna no tendría sentido que el insurgente contactara con el capitán. La línea 10305 examina el segundo elemento de la matriz de provisiones, OA(2), que corresponde a las armas, y si éste es igual a 0, no habrá, por supuesto, ningún arma y se devolverá el control al programa principal. Si hubiera armas disponibles, el insurgente formulará su oferta.

Se le indicará que digite S o N a la propuesta en la línea 10326, y la línea 10330 comprobará si la entrada es correcta. Si no lo es, retrocederá y solicitará otra respuesta.

Si decide vender las armas, la rutina saltará a la línea 10400, donde finalmente se podrá desarrollar el evento. La línea 10405 genera un número aleatorio entre 0 y 1, proporcionando un 75 % de probabilidades de enviar el programa a la línea 10450, que hará fracasar la rebelión. Si usted cree que las probabilidades en contra de que triunfe la revuelta son demasiado elevadas, puede variarlas para adecuarlas mejor a su sentido de la equidad.

Si la revuelta fracasa, el jefe se apropiará de las mercancías e incendiará el barco, con lo cual el juego habrá terminado. De lo contrario, el programa continuará a través de la línea 10429, que asegura el triunfo de la rebelión, y el nuevo jefe recientemente proclamado aprovisionará por completo el barco antes de que éste vuelva a hacerse a la mar.

La ecuación de la línea 10429 calcula la cantidad de perlas obtenidas en el intercambio, multiplicando la cantidad de armas, OA(2), por 30, sumándole el total subsiguiente al primer elemento de la matriz AO() para registrar la cantidad de perlas adquiridas. La línea 10430 suprime las armas de la matriz de provisiones. A los fines del juego no es estrictamente necesario hacer esto, pero es un buen hábito de programación y hará que resulte más sencillo ampliar el juego para incluir, por ejemplo, incidentes durante el viaje de regreso.

Si usted se niega a aprovechar el ofrecimiento, el jefe actual podría recompensarlo con 50 perlas. Esto se decide al azar en la línea 10345. La subrutina final comienza en la línea 10500 y se ocupa del viaje de regreso y la venta de las mercancías.

Complementos al BASIC

Spectrum

Reemplazar AO() por E(), V1() por B() y V2() por D() en todo el listado e introducir estos cambios:

```
10310 CLS:GO SUB 9200
10328 INPUT I$:LET I$=I$(TO 1)
10354 LET I$=INKEY$:IF I$=""THEN GO TO
      10354
10400 CLS:GO SUB 9200
10501 CLS:GO SUB 9200
10547 LET I$=INKEY$:IF I$=""THEN GO TO
      10547
```

BBC Micro:

Introducir las siguientes modificaciones:

```
10310 CLS: GO SUB 9200
10354 I$=GET$
10400 CLS:GOSUB 9200
10501 CLS:GOSUB 9200
10547 I$=GET$
```

Módulo 13: Fin del viaje

Adición al cuerpo principal del programa

```
893 GOSUB 10300
894 GOSUB 10500
```

Rutina Revuelta

```
10300 REM REVOLUCION
10305 IF OA(2)=0 THEN RETURN
10310 PRINT CHR$(147):GOSUB 9200
10315 S$="DURANTE LA NOCHE UN RIVAL DEL*":GOSUB 9100
10316 S$="JEFE VISITA EL BARCO EN SECRETO*":GOSUB 9100
10317 PRINT:GOSUB 9200
10318 S$="QUIERE COMPRAR TUS ARMAS*":GOSUB 9100
10320 S$="PARA UNA REVUELTA*":GOSUB 9100
10322 PRINT:GOSUB 9200
10324 S$="TE OFRECE 30 PERLAS POR CADA ARMA*":GOSUB 9100
10326 S$="LE VENDES LAS ARMAS?(S/N)*":GOSUB 9100
10328 INPUT I$:I$=LEFT$(I$,1)
10330 IF I$<>"N"AND I$<>"S" THEN 10328
10332 IF I$="S" THEN 10400
10334 PRINT:GOSUB 9200
10336 S$="EL JEFE SE ENTERA Y SE SIENTE*":GOSUB 9100
10338 S$="AGRADECIDO HACIA TI*":GOSUB 9100
10340 S$="TE DA PROVISIONES GRATIS*":GOSUB 9100
10342 S$="PARA EL VIAJE DE REGRESO*":GOSUB 9100
10344 GOSUB 9200
10345 IF RND(1)<.75 THEN 10350
10346 S$="Y 50 PERLAS!!*":GOSUB 9100
10348 AO(1)=AO(1)+50
10350 PRINT:GOSUB 9200
10352 S$=K$:GOSUB 9100
10354 GET I$:IF I$=""THEN 10354
10359 RETURN
10400 PRINT CHR$(147):GOSUB 9200
10405 IF RND(1)<.75 THEN 10450
10410 S$="LA REVUELTA HA TRIUNFADO*":GOSUB 9100
10412 PRINT:GOSUB 9200
10415 S$="EL NUEVO JEFE TE RECOMPENSA CON*":GOSUB 9100
10420 S$="PROVISIONES GRATUITAS PARA EL VIAJE*":GOSUB 9100
10425 S$="DE REGRESO.*":GOSUB 9100
10429 AO(1)=AO(1)+(OA(2)*30):REM SUMAR PERLAS
10430 OA(2)=0
10431 GOTO 10350
10450 S$="LA REVUELTA FRACASA!!*":GOSUB 9100
10452 PRINT:GOSUB 9200
10455 S$="EL VIEJO JEFE ESTA INDIGNADO CONTIGO*":GOSUB 9100
10457 S$="TE QUEMA EL BARCO Y ROBA*":GOSUB 9100
10458 S$="TODO!!*":GOSUB 9100
10459 PRINT:GOSUB 9200
10460 S$="     JUEGO TERMINADO!!     ":GOSUB 9100
10462 END
10464 GOTO 10462
```

Rutina Fin del viaje

```
10500 REM FIN DEL VIAJE
10501 PRINTCHR$(147):GOSUB 9200
10505 S$="CON UN TRIPULACION FUERTE Y VIENTOS*":GOSUB 9100
10507 S$="FAVORABLES EL VIAJE DE REGRESO VA*":GOSUB 9100
10508 S$="BIEN Y DURA SOLO 8 SEMANAS*":GOSUB 9100
10512 WW=0
10514 FOR T=1 TO 5
10516 WW=WW+(8*CC(T)*WG(T))
10518 NEXT
10519 PRINT:GOSUB 9200
10520 S$="FACTURA SALARIAL PARA EL VIAJE DE REGRESO=*":GOSUB
      9100
10522 PRINT WW;"PIEZAS DE ORO"
10524 PRINT:GOSUB 9200
10526 S$="CUANDO REGRESAS*":GOSUB 9100
10528 S$="PARA VENDER TUS MERCANCIAS*":GOSUB 9100
10530 S$="ESTAS VALEN ENTONCES:*":GOSUB 9100
10532 PRINT"PERLAS-";V2(1);"PIEZAS DE ORO"
10534 PRINT"FIGURILLAS-";V2(2);"PIEZAS DE ORO"
10536 PRINT"ESPECIAS-";V2(3);"PIEZAS DE ORO"
10538 PRINT:S$="OBTIENES UN TOTAL DE*":GOSUB 9100
10540 X=(AO(1)*V2(1))+(AO(2)*V2(2))+(AO(3)*(V2(3))
10542 PRINT X;"PIEZAS DE ORO"
10545 PRINT:S$=K$:GOSUB 9100:PRINT
10547 GET I$:IF I$="" THEN 10547
10550 S$="AHORA POSEES:*":GOSUB 9100
10552 PRINT MO+X;"PIEZAS DE ORO"
10555 PRINT:GOSUB 9200
10556 S$="LA FACTURA SALARIAL PARA EL VIAJE ES DE*":GOSUB 9100
10557 PRINT WT+WW;"PIEZAS DE ORO"
10559 PRINT:GOSUB 9200
10560 S$="TERMINAS EL VIAJE CON:*":GOSUB 9100
10562 Z=MO+X-WT-WW
10565 PRINT Z;"PIEZAS DE ORO"
10566 PRINT:GOSUB 9200
10567 PRINT:S$="TU CLASIFICACION ES:*":GOSUB 9100:PRINT
10568 IF Z>3200 THEN S$="CAPITALISTA DE PRIMER ORDEN*":GOSUB
      9100:END
10569 IF Z>2500 THEN S$="INSIGNE COMERCIANTE*":GOSUB 9100:END
10570 IF Z>2000 THEN S$="MERCACHIFLE DE III CLASE*":GOSUB 9100:END
10571 IF Z>1000 THEN S$="MAS BIEN UN PRIMO*":GOSUB 9100:END
10572 S$="MAS PATO QUE PIRATA"
10573 GOSUB 9100:END
```

El viaje de regreso dura ocho semanas y se debe calcular la factura de salarios. En la línea 10512 se crea una variable, WW, que representa la factura salarial para el viaje de regreso. Su valor se calcula en el bucle que empieza en la línea 10514. La fórmula multiplica el contador de cada categoría de tripulante, CC(T), por el salario semanal para cada uno, WG(T), para producir la factura salarial semanal para todos los tripulantes de esa categoría. El contador del bucle T asegura que este proceso se repita para cada categoría de tripulante, produciendo finalmente una factura de salarios semanal total para toda la tripulación. Para calcular la factura del viaje de regreso, este nuevo total se multiplicará por 8, y el programa imprimirá la factura total para la travesía de retorno.

Cuando el barco llegue a puerto se venderán las mercancías. Sus valores de mercado se establecen mediante la matriz V2(3) DIMensionada en la línea 62. Estos valores se establecieron al azar al principio del programa. Las líneas 10532-10536 de la rutina fin del viaje visualizan los valores retenidos en esta matriz, para cada uno de los tipos de mercancías comerciadas.

La cantidad total de oro recibida por las mercancías comerciadas se calcula mediante la línea 10540 y se almacena en X. La fórmula utilizada para este cálculo multiplica la cantidad de cada artículo, registrada en la matriz AO(), por el valor de mercado actual almacenado en V2(). Se suman los ingresos por las perlas, figurillas y especias y se imprime el total.

Es posible que al inicio del viaje no se haya gastado todo el capital original de 2 000 piezas de oro. Cualquier saldo se conservará a lo largo del viaje en la variable MO y se sumará al oro recibido durante la venta; la cantidad se visualizará mediante la línea 10552. La factura salarial total para el viaje se calcula en la línea 10557 sumándole la factura salarial para el viaje de ida, WI, a la factura para el viaje de regreso. El programa calcula las ganancias comerciales restando la factura salarial total de la cantidad de oro que haya en las arcas. Por último, se juzga lo bien que se ha desempeñado usted produciendo una clasificación basada en el monto de los beneficios que haya obtenido.



Ian McKinnell

El mejor lugar

Iniciamos una serie dedicada a analizar en detalle el sistema operativo del Sinclair Spectrum. Examinaremos, en primer lugar, el mapa de memoria

A diferencia del BBC Micro, el Spectrum de Sinclair no tiene una ROM dentro del ordenador reservada para el sistema operativo, puesto que un mismo chip alberga el intérprete del BASIC y el OS. Más aún, sólo existe una versión de esta ROM y el chip que la alberga es el mismo tanto en el Spectrum como en el Spectrum+, por lo que no se pueden esperar muchos cambios en el futuro. El OS es un conjunto orgánico de rutinas (de un tamaño la mitad que el OS del BBC) que ocupa tan sólo 7 K del chip del Spectrum que contiene la ROM de 16 Kbytes.

A pesar de la falta de variantes y de la naturaleza compacta de la codificación, el OS del Spectrum presenta una desventaja respecto del OS del BBC Micro. Salvo mínimas excepciones, no está vectorizado, por lo que resulta más difícil alterar el tratamiento que hace el OS de ciertos eventos. Difícil pero no imposible, como tendremos oportunidad de observar.

Suponiendo que usted estudió los capítulos referentes al OS del BBC Micro, en este momento tendrá una cierta familiaridad sobre el papel desempeñado por el sistema operativo en un ordenador. Como breve resumen, le recordamos que el OS se ocupa de todas aquellas rutinas de operaciones de entradas, salidas, visualización y almacenamiento en ficheros, interrupciones, y, en general, de toda operación que haga de puente entre el hardware y el programa del usuario o el intérprete del BASIC. Comencemos analizando el mapa de memoria, que vemos ilustrado a la derecha.

El primer bloque de 16 K de la memoria está ocupado por un chip de ROM que contiene el intérprete del BASIC y el sistema operativo. Grosso modo, el OS ocupa los 6 o 7 K inferiores de la ROM, y el intérprete, la parte superior restante. Cuando se emplea una Interface 1, los 8 K inferiores de la memoria se distribuyen en páginas de un modo similar a la paginación ya vista en el BBC Micro realizada en el área de memoria que va desde &8000 al &BFFF. Cuando se desea acceder a alguna de las facilidades que ofrece la Interface 1 (microdrives, interface serial, red de área local) la ROM normal queda "despaginada" y la ROM de la Interface 1 "paginada". Pero de esto hablaremos detalladamente más adelante.

El resto del espacio de la memoria lo ocupa la RAM, y es de notar que existen muy pocas diferen-

Datos internos
Estos mapas de memoria de un Spectrum (o de un Spectrum+) ilustran cómo afecta a ciertas áreas de la RAM el acoplamiento de una Interface 1. El área para variables de sistema se extiende normalmente hasta 23733, pero supera esta posición cuando existe una Interface 1

SPECTRUM
65536
65306 Gráficos defin. usuario
Espacio libre
Pila GOSUB
Pila Z-80
Pila calculadora
Espacio de trabajo de entrada y edición
Variables
Espacio programa
Datos canal
23734 Variables sistema
23552 Buffer impresora
23295 Archivo atributos
22528 Archivo visualización
16384
16383 Intérprete del OS y del BASIC
0

RAMTOP
STKEND
ELINE
VARS
PROG
CHANS
PROG
CHANS

SPECTRUM CON INTERFACE 1
Gráficos defin. usuario
Espacio libre
Pila GOSUB
Pila Z-80
Pila calculadora
Espacio de trabajo de entrada y edición
Variables
Espacio programa
Datos canal
Mapas microdrive
Variables sistema ROM de Interface 1
Variables sistema
Buffer impresora
Archivo atributos
Archivo visualización
Intérprete del OS y del BASIC
ROM de la Interface 1, intérprete de OS y del BASIC

Liz Dixon



cias en el empleo de la RAM de un Spectrum cuando se está utilizando la Interface 1.

Si examinamos cada una de las secciones del mapa, el *archivo de la visualización (display file)* almacena información sobre la forma de las figuras que han de visualizarse en la pantalla. Los datos referentes al color, BRIGHT y FLASH no están en este archivo, sino que se almacenan en el *fichero de atributos (attribute file)*, ya que esta información corresponde a un cuadrado de carácter en la pantalla. El *buffer para impresora (printer buffer)* es un área de 256 bytes que requiere la impresora ZX cuando se desea una copia sobre papel de la pantalla.

Las *variables del sistema* son empleadas por el OS y por el intérprete, de modo que puedan inspeccionar lo que está sucediendo en el sistema. Más adelante estudiaremos algunas de las variables del sistema más imprescindibles. En el mismo manual para usuarios del Spectrum se encuentra una lista completa de tales variables. Lista que, por desgracia, falta en el manual del Spectrum+.

Aquí es donde se hacen mucho más evidentes las diferencias entre un Spectrum con o sin Interface 1. Es evidente que la ROM de la Interface 1 necesita un espacio de trabajo y variables del sistema para sus tareas, de modo que cuando está siendo empleada genera algunas variables extra, que son colocadas a continuación de las variables del sistema normales.

Esto produce un efecto de "corrimiento" de toda la memoria empleada, con lo cual hay que realizar ciertos reajustes en consonancia.

Los *mapas para microdrive (microdrive maps)* son asimismo típicos de la Interface 1, empleados tan sólo cuando ésta vigila las áreas ocupadas y no ocupadas de la cinta del microdrive.

El área para *datos de canal (channel data)* nos ofrece una forma reducida de vectorización para las instrucciones LLIST, LPRINT, INPUT# y PRINT#. Así, por ejemplo, podemos escribir código máquina para dirigir una impresora corriente y alterar los datos de canal para que cuando encuentre la instrucción LPRINT o LLIST obedezca a nuestro código máquina en vez de las habituales rutinas del intérprete del BASIC. También de esto hablaremos con mayor extensión.

Los datos de canal también son empleados por la Interface 1.

Llegamos ahora al espacio de trabajo del programa en BASIC y de las variables, que el sistema emplea para almacenar el texto de un programa en BASIC y las variables requeridas por éste.

Los espacios de trabajo para edición *(edit)* y entrada *(input)* de datos sirven para obtener mediante EDIT una línea entera de programa o para entrar datos y líneas de programas. Una vez finalizada la edición o la entrada del texto, las líneas de programa se almacenan en su lugar específico dentro del programa, quedando libres estas áreas de nuevo.

Encontramos después un grupo de pilas. La *pila calculadora* sirve al intérprete del BASIC para sus operaciones aritméticas. La *pila Z80* es utilizada por la CPU de modo muy semejante a como el 6502 emplea la página uno de su memoria, es decir, le sirve para almacenar direcciones de retorno con subrutinas en código máquina y datos que han sido llevados a la pila para su almacenamiento temporal. La *pila* GOSUB es otra de las partes de la memoria que utiliza el intérprete del BASIC. Almacena las direcciones de retorno en las sentencias GOSUB del BASIC.

Por último, el área para gráficos *definidos por el usuario* retiene los datos de los caracteres que fueron, claro está, definidos por éste.

Dónde colocar el código máquina

De cuanto se ha dicho hemos de concluir que no hay reservado en el mapa de la memoria espacio alguno para los programas en código máquina. Es lo que sucede también en el BBC Micro si se le acopla una unidad de disco o una interface Econet. Tres son las áreas posibles para colocar en el Spectrum el código máquina. Están en una sentencia REM de línea 1 (se puede averiguar la posición de la línea 1 en la memoria a través del valor retenido en la variable del sistema PROG). Si a este valor sumamos cinco obtenemos la dirección del primer carácter en una sentencia REM de línea 1.

Otras áreas son el buffer para impresora y entre RAMTOP (parte superior de la RAM) y la parte de memoria usada para almacenar datos de gráficos definidos por el usuario. Este último método es similar al empleo de la memoria entre HIMEM y el inicio de la RAM de video en el BBC Micro.

El cuadro que proporcionarnos a continuación compara los métodos entre sí:

Sitio	Spectrum	Spectrum e Interface 1
REM línea 1	✔	✘
Buffer impresora	(✔)*	(✔)*
Encima de RAMTOP	✔	✔

* No utilizable cuando se emplea una impresora

Vemos que sólo la última posibilidad es utilizable en la práctica, dado que los restantes métodos son arriesgados en determinadas circunstancias. Lo que se encuentra por encima de RAMTOP queda a salvo de una posible sobreescritura del BASIC, además de ser fácil la reserva de espacio. La posición exacta de RAMTOP se retiene como una variable de dos bytes en las direcciones 23730 y 23731. El valor se almacena en el formato corriente del Z80 *(lo-hi)*. Para comprobar la posición actual de RAMTOP haga lo siguiente:

```
LET ramtop=PEEK23730+256*PEEK23731
```

Una vez anotada la posición de RAMTOP (que suele ser 65306 en un ordenador de 48 K "limpio"), se habrán de calcular cuántos bytes se necesitan para el programa en código máquina. Después de calcular el valor que se obtiene mediante la fórmula RAMTOP−(n.º de bytes en el programa+1), se bajará RAMTOP con la instrucción CLEAR:

```
CLEAR valor
```

Esta instrucción, en un programa en BASIC, realizará un CLS, repondrá la posición de PLOT y ejecutará un RESTORE. Limpiará, además, y reubicará la pila GOSUB. Por esta razón habrá de habituarse a emplear con cuidado la instrucción y a colocarla en lo posible al principio del programa, de lo contrario no evitará los problemas.

Después de ejecutar la instrucción CLEAR valor, la

nueva posición de RAMTOP estará almacenada en 23730 y 23731, y el primer byte disponible para su código máquina se encontrará en la dirección (valor+1).

Así, suponiendo una máquina de 48 K limpia, la orden CLEAR 59999 reservará unos 5 K de memoria para sus programas, comenzando por la posición 60000.

Tras la reserva de memoria anterior, la instrucción POKE puede servir para almacenar los bytes que componen su programa en la memoria.

Variables del sistema

Finalizaremos esta introducción sobre el OS del Spectrum repasando algunas variables del sistema que contienen información sobre el modo como la memoria está distribuida en el sistema. El acceso a dichas variables ha de realizarse mediante PEEK o bien POKE referidas a las posiciones adecuadas, pues no existen equivalentes de OSBYTE u OSWORD, llamadas que analizamos detalladamente al estudiar el sistema operativo del BBC Micro.

- chars: Esta variable, situada en 23606 y 23607, apunta a una posición situada 256 bytes más abajo de la información del conjunto de caracteres. Habitualmente la dirección que alberga esta variable de dos bytes apunta a la dirección que está en la ROM 256 bytes más abajo respecto a la dirección del primer byte de la definición del carácter espacio (CHR$32).

La dirección actualmente albergada en la variable puede conocerse, como la de las restantes, de la siguiente manera:

```
LET chars=PEEK 23606+256*PEEK 23607
```

Si se desea, es posible alterar el valor contenido en la variable para que apunte a la información que defina un nuevo conjunto de caracteres, que puede almacenarse en la RAM.

- vars: Se encuentra en las posiciones 23627 y 23628, y contiene la dirección de inicio de las variables del BASIC. Se pueden inspeccionar (PEEK) estos valores pero no alterarlos, ya que "despistaría" al Spectrum sobre sus propias variables. La línea:

```
LET vars=PEEK 23627+256*PEEK 23628
```

le proporciona la dirección de inicio de las variables.

- prog: Contiene la dirección de inicio del programa en BASIC.

Es, pues, equivalente a la variable de sistema PAGE en el BBC Micro. Se halla en 23635 y 23636, por lo que si se desea conocer la dirección del comienzo del programa, haremos

```
LET prog=PEEK 23635+256*PEEK 23636
```

Es de notar que en este caso

```
PRINT vars-prog
```

nos dará la longitud del programa.

- eline: Se encuentra en 23641 y 23642. Contiene el inicio de cualquier texto entrado en el sistema. Se puede emplear para conocer la cantidad de memoria ocupada por las variables empleadas. Suponiendo calculada vars, escribiríamos

```
LET eline=PEEK 23641+256*PEEK 23642
PRINT eline-vars
```

para obtener dicha cantidad de memoria.

- chans: Contiene la dirección del inicio de la información del canal. Situada en 23631 y 23632, estudiaremos con más detalle el tema de los canales en próximos capítulos.

- stkend: Se encuentra en las posiciones 23653 y 23654. Junto con el valor de RAMTOP, nos permite saber la cantidad de memoria libre para nuestro programa en BASIC y sus variables. Escribiremos, para obtener el valor de stkend:

```
LET stkend=PEEK 23653+256*PEEK 23654
```

Y para conocer la cantidad de memoria que queda:

```
PRINT ramtop-stkend
```

También es posible conocer dicha cantidad mediante una rutina de la ROM. Pero este método no es tan exacto como el que acabamos de proponer. Suponiendo calculado el valor de RAMTOP:

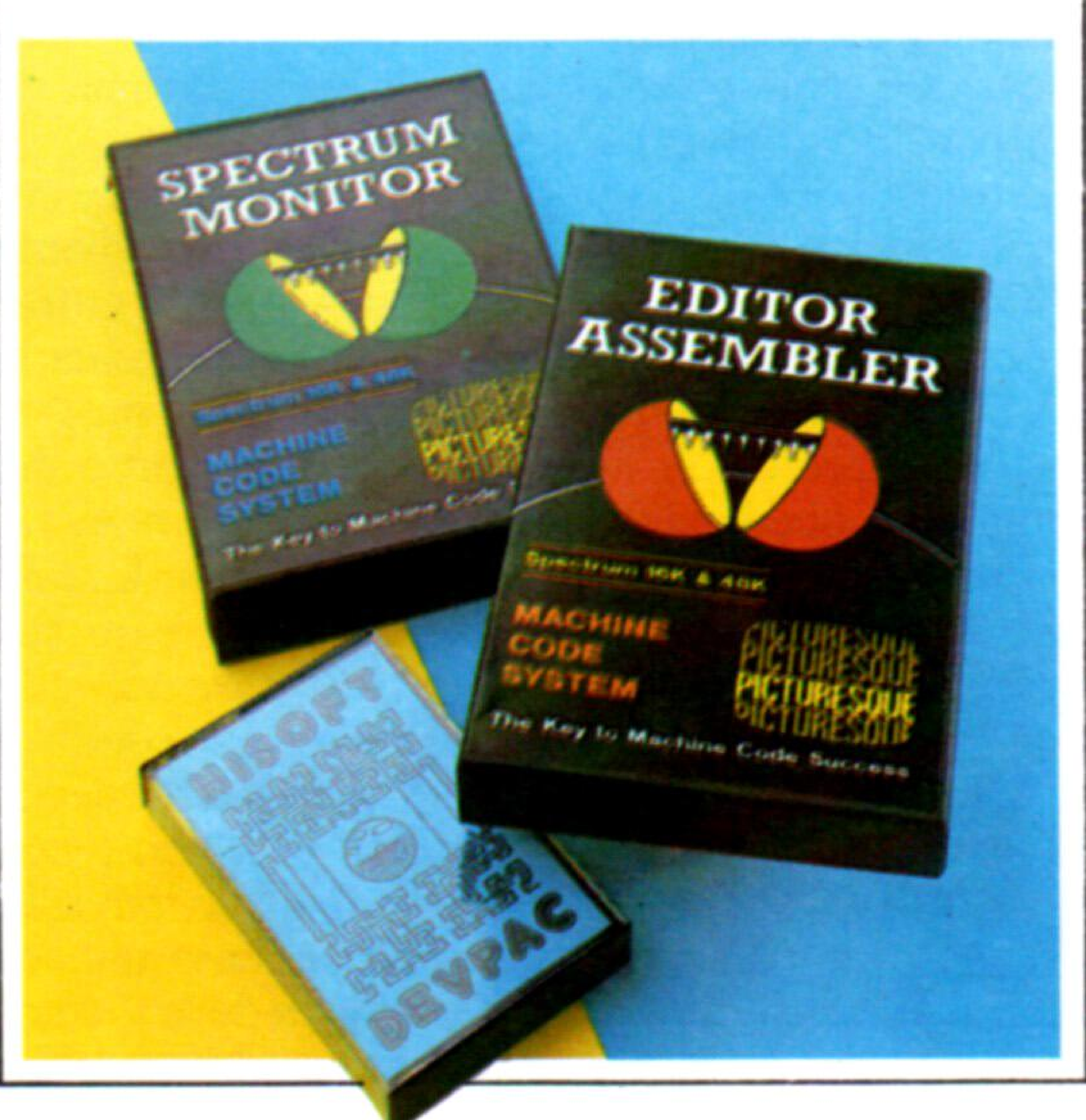
```
PRINT ramtop-USR 7962
```

nos calculará también el espacio libre.

Hora del assembly

Hay varios paquetes ensambladores para el usuario del Spectrum interesado en profundizar en el código máquina. Uno de los más conocidos es DEVPAC 3, de Hisoft. Distribuido en cassette, incluye el ensamblador GENS3 y el monitor/desensamblador MON3. Existe también la versión para microdrive. Aunque su empleo está muy extendido, no es de los más cómodos del mercado, aunque también es verdad que resulta el más barato.

El Editor Assembler y el Spectrum Monitor (que incluye un desensamblador), de Picturesque, es un paquete doble, el primero a un precio algo más elevado que el segundo. Una de las principales ventajas del sistema de Picturesque es que se puede acoplar muy bien a un Spectrum de 16 Kbytes. Los manuales que acompañan a este paquete superan al de Hisoft, y el conjunto tiene un carácter mucho más profesional



Chris Stevens

Vara para medir

Una facilidad ausente en el Spectrum es el medio para calcular la longitud de un programa; esto se consigue con PEEK, pero aquí va una pequeña utilidad que se sirve de las variables PROG y VARS para calcular la longitud de cualquier programa en BASIC actualmente en la memoria, y la almacena en la REM de línea 1. Sólo exige que la primera línea del programa sea una sentencia REM de al menos cinco caracteres, como, por ejemplo:

```
1 REM 00000 bytes de largo
```

cuando se ejecuta la rutina en código máquina, la sentencia REM sufre una modificación para contener la longitud del programa. Por ejemplo:

```
1 REM 00801 bytes de largo
```

Esto es bastante .ítil, ya que a la siguiente vez que se examine el programa tendremos presente su longitud. El primer fragmento de código es un listado en assembly del código máquina en cuestión. Si no posee un ensamblador conveniente, el segundo listado está en BASIC y es un programa utilizable en la carga del programa en código máquina. En el Cargador de BASIC, CLEAR 61999 da ese número como valor de la variable de sistema RAMTOP. Por tanto, el primer byte "libre" se encuentra en la dirección 62000, que hemos empleado como dirección de inicio de la rutina en código máquina. RAMTOP tendrá este valor hasta que no venga alterado con una instrucción CLEAR n,o bien se ejecute una PRINT USR 0 e incluso si el ordenador se apaga y vuelve a encender. Esto quiere decir que NEW no afectará al valor de REMTOP, además de que la rutina en código máquina está protegida de cualquier sobreescritura en BASIC mientras RAMTOP quede inalterada. Una vez que el programa está en la memoria, se emplea USR para ejecutar el código máquina. El argumento de la función USR (n) es la dirección de la rutina en lenguaje máquina que desea ejecutar. Tal dirección puede estar en cualquier sitio en la memoria. Así, lo siguiente ejecutará el código máquina en la dirección 62000, y cualquiera puede ser usado para ejecutar el programa en código máquina que acabamos de ver:

```
RANDOMISE USR (62000)
PRINT USER (62000)
LET L=USR (62000)
```

Los métodos con el PRINT y el LET con USR dan también resultados utilizables por el BASIC. El resultado es el valor contenido en el par de registros BC cuando la rutina en código máquina termina y retorna al BASIC; así, una rutina en código máquina puede devolver los resultados al BASIC. Si su rutina no afecta al valor del registro BC, lo que se obtiene es la dirección de la rutina.

Listado assembly

```
62000                          ORG    62000
62000  2A4B5C                  LD     HL,(23627)   toma VARS en HL
62003  ED5B535C                LD     DE,(23635)   toma PROG en DE
62007  AF                      XOR    A            limpia flag arrastre
62008  ED52                    SBC    HL,DE        pone longitud progr. en HL
62010  DD2A535C                LD     IX,(23635)   inicio programa en IX
62014  DD23                    INC    IX
62016  DD23                    INC    IX           incrementa IX para que apunte
62018  DD23                    INC    IX           al carácter post. al token REM
62020  DD23                    INC    IX           de la primera línea del prog.
62022  DD23                    INC    IX
62024  011027                  LD     BC,10000     inicio toma códigos ASCII
62027  CD6BF2                  CALL   PRINT        de cada dígito de la long. del
62030  DD23                    INC    IX           programa y lo almacena en
62032  01E803                  LD     BC,1000      la sentencia REM
62035  CD6BF2                  CALL   PRINT
62038  DD23                    INC    IX
62040  016400                  LD     BC,100
62043  CD6BF2                  CALL   PRINT
62046  DD23                    INC    IX
62048  010A00                  LD     BC,10
62051  CD6BF2                  CALL   PRINT
62054  DD23                    INC    IX
62056  010100                  LD     BC,1
62059  AF            PRINT:    XOR    A            limpia cont. arrastre y cero
62060  B7            L1:       OR     A
62061  ED42                    SBC    HL,BC        resta potencia de diez de
                                                   la long. del programa...
62063  3803                    JR     C,FINISH     y salta si es negativo
62065  3C                      INC    A            si no lo es incrementa A
62066  18F8                    JR     L1           repetición
62068  09                FINISH:ADD   HL,BC        repone la long. del prog.
                                                   a valor positivo para
                                                   la siguiente potencia de diez
62069  C630                    ADD    A,48         toma el código ASCII del dígito
62071  DD7700                  LD     (IX),A       lo almacena en REM
62074  C9                      RET                 fin
```

Cargador del BASIC

```
  10 REM 00000 bytes de largo
  20 CLEAR 61999
  30 FOR i=0 TO 74
  40 READ a: POKE 62000+i,a
  50 NEXT i
1000 DATA 42, 75, 92, 237, 91, 83, 92, 175, 237, 82, 221, 42, 83, 92, 221, 35,
     221, 35, 221, 35, 221, 35, 221, 35, 1, 16, 39, 205, 107, 242, 221, 35, 1,
     232, 3, 205, 107, 242, 221, 35, 1, 100, 0
1010 DATA 205, 107, 242, 221, 35, 1, 10, 0, 205, 107, 242, 221, 35, 1, 1, 00,
     175, 183, 237, 66, 56, 3, 60, 24, −8, 9, 198, 48, 221, 119, 0, 201
```

Nota: Para asegurarse de que esta *Vara para medir* localiza la posición correcta para los dígitos indicadores de la longitud del programa, no se inserten espacios entre el número de la primera línea y la instrucción REM (fuera del espacio automático que prevé el sistema del Spectrum)

De la TV a la VDU

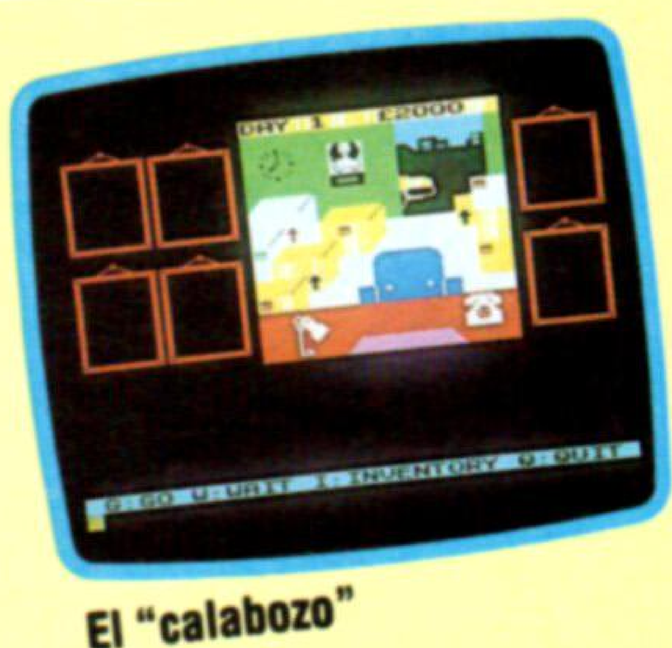
Haciendo su agosto
Éstas son cuatro escenas tomadas de *Minder*. El objetivo del juego es vender su propio stock (almacenado en su "calabozo") a los comerciantes. Si usted se queda sin existencias, puede adquirir otras mercancías a través de los clientes del bar Winchester. Por supuesto, debe transar para conseguir el mejor precio posible, tanto a la hora de comprar como de vender, con el fin de ganar la máxima cantidad de dinero en el tiempo de que dispone



El bar Winchester



Haciendo un negocio



El "calabozo"



Lista de existencias

"Minder" (Guardián), programa producido por la firma Euston Films, es un original y entretenido juego inspirado en una popular serie de la televisión británica

El programa de televisión *Minder* se ha convertido en uno de los mayores éxitos de la televisión británica de los últimos años, y su conversión al formato de juego quizás haya llegado con demasiado retraso (la serie comenzó a emitirse a mediados de los setenta).

Minder se basa en los oscuros asuntos de Arthur Daley, conocido traficante de mercancías dudosas de todo tipo, y Terry McCann, su *minder* (guardián) y secuaz. En este juego usted representa el papel de Arthur, y el objetivo consiste en ganar el máximo dinero posible en quince días mediante la compra y venta de mercancías. Con este fin, debe contactar con otros comerciantes a quienes vender su mercancía. Con frecuencia tendrá cosas bastante peculiares que vender, como latas de subproductos animales o "ferkinators nucleónicos". Con igual frecuencia, los comerciantes le ofrecerán una cantidad demasiado reducida por los artículos y usted habrá de regatear con ellos para llegar a un acuerdo más razonable.

Una de las mayores espinas de Arthur es el sargento-detective Chisholm, oficial del Departamento de Investigación Criminal local. La influencia de Chisholm comienza cuando un comerciante interesado en comprarle algunos ordenadores personales, que *por casualidad* usted ha adquirido, le informa que el sargento-detective en realidad está a la caza de ordenadores robados. Estas noticias, por supuesto, hacen que el precio descienda.

Los problemas de Arthur se agravan aún más si Chisholm se presenta en el "calabozo" (*lock-up*: el almacén donde Arthur guarda sus mercancías) y encuentra los artículos robados.

Para hacer negocios y encontrar a Terry, a quien usted necesita para transportar sus mercancías desde y hasta el calabozo, por lo general habrá de acudir a Winchester, un club de bebedores. En este escenario, como en todos los otros del juego, hay hasta seis personajes con los que puede hablar. Sus rostros aparecerán en uno de los marcos de alrededor de la pantalla, y para abordar a uno de ellos usted simplemente pulsa la tecla correspondiente a ese personaje.

Abordar a los personajes de Winchester no es realmente necesario, a menos que desee hablar con alguno en particular, como Dave, el barman, o Terry; la mayoría de los demás intentarán venderle mercancías tan valiosas como jaulas de caoba para conejos o trajes de polietileno para la lluvia.

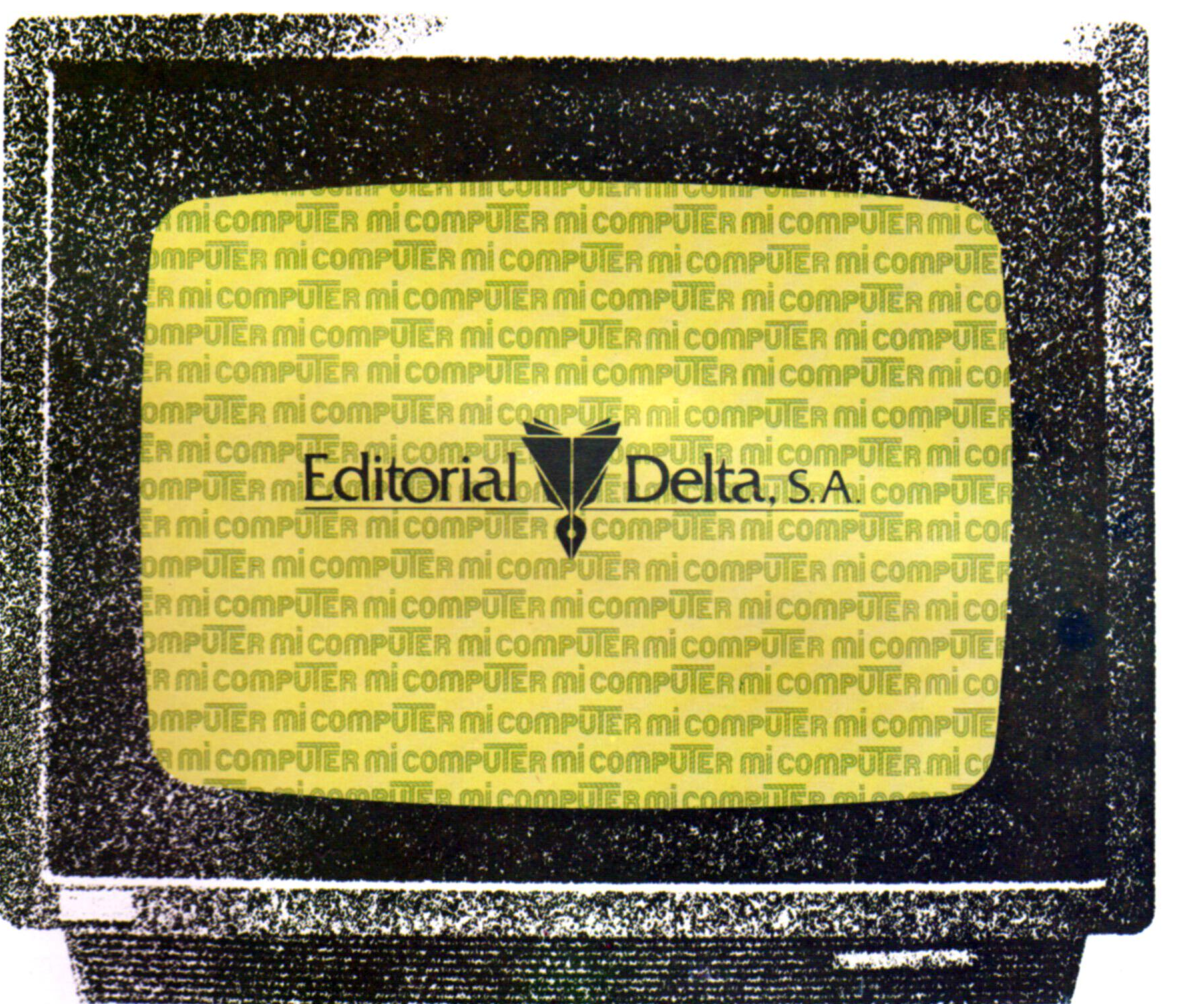
Aquí la estrategia es muy similar a cuando se trata de vender. Primero debe averiguar a cuánto está vendiendo estos artículos su contacto y qué cantidad de los mismos posee, y después puede empezar a regatear. Mientras negocia el precio, un reloj va marcando el tiempo: el contacto se marchará si el trato no se cierra dentro de un cierto margen. Si usted decide que ya no quiere regatear más, puede digitar BYE y esa persona desaparecerá.

Terry es un personaje algo esquivo y quizá no pueda establecer contacto con él durante algún tiempo, y cuando termina una tarea espera un favor a cambio: un trago o un premio. Por consiguiente, cuando negocie un trato debe tener en cuenta que de los beneficios habrá de extraer la paga de Terry.

Sin duda alguna, la parte más divertida de este juego es la negociación. Mientras usted y su oponente regatean el precio, la pantalla visualizará una serie de comentarios como "fantástica calidad" o "pequeño y simpático negociante". Cuando usted ofrece un precio, es importante que lo digite con la sintaxis correcta, como "Estoy ofreciendo 200 pesetas". De lo contrario, quizá el ordenador no le entienda.

A diferencia de muchos juegos de esta clase, *Minder* es tan entretenido e interesante como la serie de televisión. Consigue captar con exactitud la tónica general del programa y se hará popular tanto entre los amantes de los juegos como entre quienes conozcan la serie de televisión.

Minder: Para el Sinclair Spectrum, MSX, Memotech, Amstrad y Commodore 64
Editado por: DK'tronics, Shire Hill Industrial Estate, Saffron Walden, Essex CB11 3AQ, Gran Bretaña
Autor: Don Priestley
Formato: Cassette
Palanca de mando: No es necesaria


Editorial Delta, S.A.



10089
9 788485 822836